Universidade Federal de Santa Catarina
Curso de Pós-Graduação em Ciência da Computação

Viviane Duarte Bonfim

Tratamento de Documentos Textuais Estruturados no Ambiente SEA

Florianópolis, fevereiro de 2004.

Universidade Federal de Santa Catarina
Curso de Pós-Graduação em Ciência da Computação

Viviane Duarte Bonfim

Tratamento de Documentos Textuais Estruturados no Ambiente SEA

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação

Prof. Dr. Ricardo Pereira e Silva
Orientador

Florianópolis, fevereiro de 2004.

Tratamento de Documentos Textuais Estruturados no Ambiente SEA

Viviane Duarte Bonfim

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação Área de Concentração Sistemas de Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

_____

Prof. Raul Wazlawick
Coordenador

_____

Prof. Ricardo Pereira e Silva
Orientador

Banca Examinadora

_____

Profª Fabiane Vavassouri

_____

Profª Cristiane Gresse Von

_____

Profª Patricia Vilain

"Descobrimos como é bom chegar quando se tem paciência e, para chegar onde quer que seja, aprendemos que não é preciso dominar a força, mas a razão. É preciso, antes de tudo, querer".

Amyr Klink

Agradecimentos

Primeiramente gostaria de agradecer aos meus pais, em especial à minha mãe, que mesmo tão longe a sinto tão perto: A vocês que me deram a dádiva mais linda do Universo, a vida. E me ensinaram a vivê-la com dignidade, não basta apenas agradecer... A vocês que me iluminaram os caminhos obscuros com amor e dedicação para que eu pudesse prosseguir sem medo... A vocês, que se doaram por inteiro e sacrificaram seus sonhos e anseios, para realizarem os meus... A vocês, o meu eterno Obrigado!

Mãezinha Querida, se não fosse por você, pelo teu imenso esforço, com certeza eu não estaria aqui! Tenho certeza que apesar de você não fazer mais parte desse mundo, você continua sempre ao nosso lado! Muito Obrigada!

Agradeço a quem muito admiro e estimo, meu orientador, Professor Ricardo Pereira e Silva pela paciência e sua imensa dedicação. Muito Obrigada!

Agradeço também aos meus colegas, Neiva, Glady, Érika, Gilson, Alexandre, Gilberto, Gláucio e Ricardo que sempre estiveram dispostos a me ajudar quando precisava. Agradeço em especial ao meu colega Roberto....sua ajuda foi indispensável.....Muito Obrigada por tudo! Gostaria de poder retribuir tudo que fez por mim!

Agradeço ao meu irmão pela confiança e pelo apoio....Razão da minha vida!

Ao meu namorado, pela paciência!

Agradeço à professora Ivone Ávila responsável pela revisão do meu texto!!! Sua ajuda foi imprescindível! Muito obrigada!!!

Agradeço alguém muito especial: Deus. O meu mais sincero agradecimento. Agradeço por tudo o que fui, o que sou e o que ainda serei.

O meu agradecimento ao meu Anjinho da Guarda. Esteve todo tempo ao meu lado, inspirando confiança, fazendo com que eu escolhesse sempre o melhor caminho.

Sumário

Lista de Siglas

API	*Application Program Intreface*
CASE	*Computer Aided Software Engineering*
CMM	*Capability Maturity Model*
CMMI	*Capability Maturity Model Integration*
HTML	*HyperText Markup Language*
ISO	*International Organization for Standartization*
KPA	*Key Process Area*
OO	Orientação a Objetos
PA	*Process Area*
RUP	*Rational Unified Process*
SEI	*Software Engineering Institute*
SGW	Sistema Gerenciador de *Workflow*
UML	*Unified Modeling Language*
XML	*Extensible Markup Language*
WFMS	*Workflow Management System*

Lista de Figuras

Lista de Tabelas

Resumo

Este trabalho propõe solução para uma deficiência encontrada em ambientes de desenvolvimento de software em geral: a falta de registro de informações textuais associadas aos produtos e processos, tais como registro de inspeção, especificação de requisitos, planos de teste, relatório de aplicação dos mesmos e registro de verificação da consistência de documentos. Esses aspectos precisam ser contemplados em processo compatível com o Capability Maturity Model Integration (CMMI). A partir do suporte ao desenvolvimento e uso de documentos provido pelo framework OCEAN*, foi desenvolvida uma extensão ao ambiente SEA, desenvolvido sob esse framework, para a criação, edição e verificação de consistência de documentos textuais estruturados, incorporáveis a projetos em conjunto com outros documentos, como diagramas UML. A idéia principal é transformar a produção manual de documentos em um processo estruturado através de uma estrutura de documentos textuais dentro do ambiente SEA e do tratamento organizado e estruturado destes documentos, o que leva ao registro mais efetivo de informações do processo de desenvolvimento de software, facilitando o controle das atividades realizadas em um processo, bem como dos documentos gráficos e textuais criados.*

Palavras-chave*: ambientes de desenvolvimento de software, framework, CMMI.*

Abstract

This work proposes a solution for a weakness found in several software development environments: the lack of textual registration related to products and processes, such as inspection registration, requirements specification, test plans, reports about the use of the product and processes and the registration of the consistency assessment of the documents. These aspects must be controlled in an environment that intends to be compatible with SEI´s Capability Maturity Model Integration (CMMI). To address this problem, it was built an extension of the SEA environment that uses the OCEAN´s framework support for the development of documents to create, edit and verify structured textual documents consistency. These documents may be attached to projects with other documents, such as UML diagrams. The main idea is to transform the manual production of documents in a structured process based on a structure of textual documents in the SEA environment and in the consistency analysis of these documents. The goal is to provide effective way to control the activities and the graphical and textual documents contained within a software development process.

Key-Words*: software development environments, framework, CMMI.*

1 Introdução

Ambientes de desenvolvimento de *software* proporcionam técnicas para a construção de *software* no aspecto tecnológico, como diagramas em UML*,* mas não disponibilizam meios para documentos textuais estruturados relativos à produção dos mesmos e para verificar a consistência das informações que estão sendo tratadas. Essa deficiência motivou o desenvolvimento deste trabalho que pretende criar *templates* para modelos de documentos e incorporá-los a um ambiente de desenvolvimento de *software*.

Para isso, considerou-se práticas de CMMI relacionadas à documentação e procurou-se utilizar essa visão para ampliar o escopo dos ambientes de desenvolvimento de *software* de modo que estes possam controlar seus documentos. Pretende-se, então, facilitar a produção e verificação de consistência de documentos textuais estruturados num ambiente de desenvolvimento de *software,* obedecendo aos requisitos de CMMI identificados em algumas *Process Area* (PA) do modelo, incorporando-os no ambiente de desenvolvimento de *software* SEA.

1.1 Motivação

Em geral, ambientes de desenvolvimento de *software* fornecem suporte ao aspecto tecnológico no desenvolvimento de um projeto, mas não apóiam os procedimentos gerenciais e, por isso, não oferecem apoio à produção de documentos textuais estruturados.

O SEA (SILVA, 2000) é um ambiente de desenvolvimento de *software* construído a partir do *framework* OCEAN (SILVA, 2000), o qual possui a finalidade de fornecer uma ampla estrutura para reutilização de suas classes ao construir ambientes com essa finalidade.

O SEA permite a utilização de abordagens orientadas a componentes e o desenvolvimento baseado em *frameworks* – permitindo reuso tanto de código quanto de projeto, utilizando as técnicas de *Unified Modeling Language* (UML), conjuntamente com a

tecnologia *Workflow* que proporciona o acompanhamento das atividades a serem realizadas (SILVA, 2000).

Documentar informações textuais, tais como registro de inspeção, especificação de requisitos, planos de testes, relatório de aplicação dos testes e registro de verificação da consistência de documentos é necessário ao processo, pois organiza as informações e fornece acompanhamento das atividades que estão sendo realizadas. Além disso, é preciso estabelecer uma estrutura para elas e mantê-las como parte do projeto, no mesmo repositório de especificações – estrutura do SEA (SILVA, 2000), permitindo, com isso, um controle mais rigoroso de sua produção e de sua qualidade.

1.2 Objetivo do Trabalho

O objetivo principal é transformar a produção manual de documentos textuais estruturados em um processo prático e organizado por meio de uma estrutura de documentos textuais a ser implantada no ambiente de desenvolvimento de *software* SEA e verificar a consistência desses documentos.

A estrutura de documentos textuais pode levar ao registro de informações realizadas durante o processo de desenvolvimento de *software*, facilitando a organização das atividades e o controle sobre os documentos criados.

Para alcançar o objetivo determinado, foi proposta a união de uma ferramenta para estrutura de documentos textuais contemplados em processo compatível com alguns dos requisitos do *Capability Maturity Model Integration* (CMMI) no ambiente SEA. Isso proporcionará algumas funcionalidades a mais do que já dispõe o ambiente, enriquecendo seu processo de desenvolvimento e, conseqüentemente, ampliando a qualidade do produto final.

1.3 Justificativa do Trabalho

O *framework* OCEAN suporta a definição de ambientes e estruturas de documentos tratadas por esses ambientes. No ambiente SEA, desenvolvido sob OCEAN são manipulados

documentos de estruturas distintas. Este, porém, antes do esforço de pesquisa aqui descrito não possuía suporte a documentos textuais.

Assim como o ambiente SEA, ambientes de desenvolvimento de *software* em geral não apóiam os procedimentos gerenciais em termos dos produtos de trabalho diretamente associados ao *software* em desenvolvimento. Portanto, são incapazes de apoiar os procedimentos gerenciais, em função de sua demanda de documentos textuais.

Visando suprir essa carência de registro de informações necessárias à documentação do processo em ambientes de desenvolvimento de *software,* em especial, no ambiente SEA, identificou-se a necessidade de introduzir neste uma estrutura de documentos textuais estruturados. Com isto, passa a ser possível o tratamento estruturado das informações textuais pelo próprio ambiente, de forma semi-automática.

A idéia de transformar a produção manual de documentos estruturados surgiu da necessidade de registrar as informações partindo da abordagem de projeto de *workflow*, pois, ao estabelecer atividades a serem realizadas no projeto, apenas as atividades associadas diretamente à construção de especificações e código possuem subsídios para registro das suas informações. Caso deva ser realizada uma inspeção em algum documento, ou uma especificação de requisitos, ou um plano de testes, por exemplo, não há recurso necessário para registrar sua documentação.

Tais informações poderiam ser registradas em um documento do *MSWord* por exemplo, e, sua construção até poderia ser semi-automatizada, mas não estaria fazendo parte de uma estrutura de projeto dentro de um ambiente de desenvolvimento de *software* e, além disso, não existiria uma verificação de consistência das informações definidas, a não ser de forma manual.

Como justificativa para a inclusão de documentos estruturados ao ambiente SEA, os aspectos citados devem ser contemplados em processo compatível com o *Capability Maturity Model Integration* (CMMI).

A escolha do Modelo de Maturidade de Capacitação Integrado justifica-se por terem sido adaptados requisitos deste modelo no ambiente SEA em uma extensão desenvolvida com

base em (SCHEIDT, 2002) na dissertação de mestrado "Introdução de suporte gerencial baseado em *workflow* e *Capability Maturity Model Integration* (CMMI) a um ambiente de desenvolvimento de *software"*.

Assim, práticas de CMMI poderão auxiliar na melhora dos processos de desenvolvimento de *software* por meio do ambiente SEA que, entre outras possibilidades, fornece condições para desenvolvimento de projetos de aplicações, *frameworks* e componentes.

1.4 Metodologia do Trabalho

Para a elaboração deste trabalho, foram estabelecidas duas etapas: levantamento bibliográfico e a implementação da proposta.

Primeiramente, é realizado estudo sobre *frameworks* para entender melhor sua construção, bem como sua estrutura e forma de utilização.

A partir disso, começa-se a estudar o *framework* OCEAN, desenvolvido durante o doutorado de (SILVA, 2000) com apresentação da tese "Suporte ao desenvolvimento e uso de *frameworks* e componentes", pois o suporte descrito no presente trabalho é uma extensão de um ambiente construído a partir desse *framework*.

Após ter adquirido conhecimento sobre o OCEAN, deu-se início ao estudo do ambiente de desenvolvimento de *software* SEA, desenvolvido por (SILVA, 2000) como mencionado, construído a partir do *framework* OCEAN e estendido por (SCHEIDT, 2002) com novas funcionalidades adicionadas ao ambiente.

Para introduzir as novas funcionalidades ao ambiente SEA, identificaram-se algumas práticas de PAs de CMMI que pudessem ser adaptadas ao objetivo do trabalho, ou seja, PAs que demandam documentação.

Após ter realizado o levantamento bibliográfico dos assuntos citados acima, buscou-se trabalhos com alguma compatibilidade com o tema.

Já com os objetivos estabelecidos, foram definidos *templates* de documentos para desenvolver o que foi proposto, de forma a atingir os objetivos. Esses *templates* são baseados no *Rational Unified Process* (KRUCHTEN, 2001).

Concluída a primeira etapa, iniciou-se a segunda parte do trabalho, isto é, a implementação do protótipo da proposta "Tratamento Estruturado de Documentos Textuais Estruturados no Ambiente SEA", buscando adicionar novas funcionalidades ao ambiente.

Para isso, criou-se uma nova especificação no ambiente SEA: Estrutura de Documentos Textuais. A finalidade era suprir carência identificada no ambiente, isto é, a falta de suporte ao registro de informações textuais estruturadas necessárias à documentação do processo.

Assim, em um estudo, foram adicionadas a essa especificação alguns *templates* de documentos: registro de inspeção, especificação de requisitos, planos de testes, relatório de aplicação dos testes e a verificação da consistência desses documentos, gerada através de ferramentas automatizadas seguindo regras de consistência apresentadas no capítulo 6. O resultado da análise de consistência é armazenado num arquivo texto.

Para a implementação da proposta, utilizou-se a linguagem *Smalltalk,* pois o ambiente SEA foi implementado nessa linguagem com o apoio da ferramenta *Visual Works* que suporta a linguagem de programação *Smalltalk*.

1.5 Estrutura da Dissertação

Os capítulos deste trabalho estão organizados da seguinte forma: O capítulo 2 apresenta alguns conceitos e características sobre *frameworks*, o *framework* OCEAN, suas funcionalidades, características e estrutura. O *framework* OCEAN é uma estrutura de classes que fornece suporte à construção de ambientes de desenvolvimento de *software*. O OCEAN permite o desenvolvimento de ambientes que manipulem diferentes estruturas de especificação e diferentes funcionalidades. Devido a sua flexibilidade, o OCEAN permite a produção e alteração de especificações e possui também suporte à criação de estruturas de

documentos, edição semântica de especificações e composição de ações de edição complexas através de ferramenta. A apresentação de um capítulo sobre *frameworks* e OCEAN justifica-se pela implementação do trabalho desenvolvido, pois todas as classes criadas, são subclasses de classes do OCEAN.

O capítulo 3 apresenta o ambiente SEA, ambiente de desenvolvimento de *software* entendido a partir do *framework* OCEAN. Do ambiente SEA, são apresentados seus conceitos, estrutura e funcionalidades, bem como conceitos de *Workflow,* devido à utilização desta ferramenta nesse ambiente de desenvolvimento, que suporta o acompanhamento das atividades a serem realizadas em um projeto (SCHEIDT, 2002). Essa descrição é necessária devido à contribuição do trabalho desenvolvido, ou seja, as funcionalidades que irão contribuir para a melhora dos processos do ambiente SEA que serão introduzidas ao ambiente.

O capítulo 4 apresenta o *Capability Maturity Model Integration* - CMMI, modelo que auxilia na qualidade do processo de *software*. O capítulo enfatiza algumas PAs de CMMI, pois os seus requisitos demandam documentação o que é tratado no trabalho desenvolvido com procedimentos que serão apresentados no capítulo 7. Adianta-se que a preocupação do trabalho é fornecer subsídios que possam auxiliar em parte do cumprimento de PAs que consistam em registrar informações textuais.

O capítulo 5 apresenta alguns trabalhos correlatos em relação ao tema estudado.

O capítulo 6 demonstra os modelos e conceitos da estrutura de documentos textuais e suas regras de consistência como parte da contribuição do trabalho.

O capítulo 7 apresenta as novas funcionalidades do SEA, descrevendo a inserção da estrutura de documentos textuais no ambiente SEA, ou seja, documentos textuais estruturados, baseados nas práticas CMMI *Staged* e *templates* adotados no *Rational Unified Process* (RUP), adicionados ao ambiente SEA. A estrutura de documentos criada irá apoiar-se no registro de informações que devem ser documentados ao longo do processo de desenvolvimento de *software,* permitindo, com isso, a melhora nos seus processos. Assim, será implantada uma estrutura de documentos textuais com um conjunto de *templates* que

representam Modelos de Documentos Textuais, os quais podem fazer parte da documentação (normalmente utilizada) necessária em processos para registro de informações.

A união da estrutura e organização de documentos textuais estruturados e atividades que podem ajudar a atingir requisitos das PAs poderá auxiliar na melhoria dos produtos de trabalho em relação ao registro das informações, como especificação de requisitos, planos de testes, aplicação de testes, registro de inspeção. Isso facilitará a organização dos documentos e um acompanhamento das informações a serem registradas permitindo a verificação de consistência desses documentos de forma semi-automatizada. Esse procedimento permitirá a otimização dos processos no ambiente SEA e auxiliará a execução de processos compatíveis com CMMI.

2 *Framework* OCEAN

Este capítulo apresenta uma breve descrição de *frameworks* orientados a objetos, e o *framework* OCEAN - sua descrição, estrutura, características e vantagens. Isso é necessário para entender o trabalho desenvolvido, devido à exploração de um ambiente de desenvolvimento de *software* construído a partir do referido *framework*.

Ressalta-se que o *framework* OCEAN foi desenvolvido durante o doutorado de (SILVA, 2000) com a tese – "*Suporte ao desenvolvimento e uso de frameworks e componentes"*. Este capítulo é, portanto, baseado nessa tese.

2.1 *Framework* Orientado a Objetos

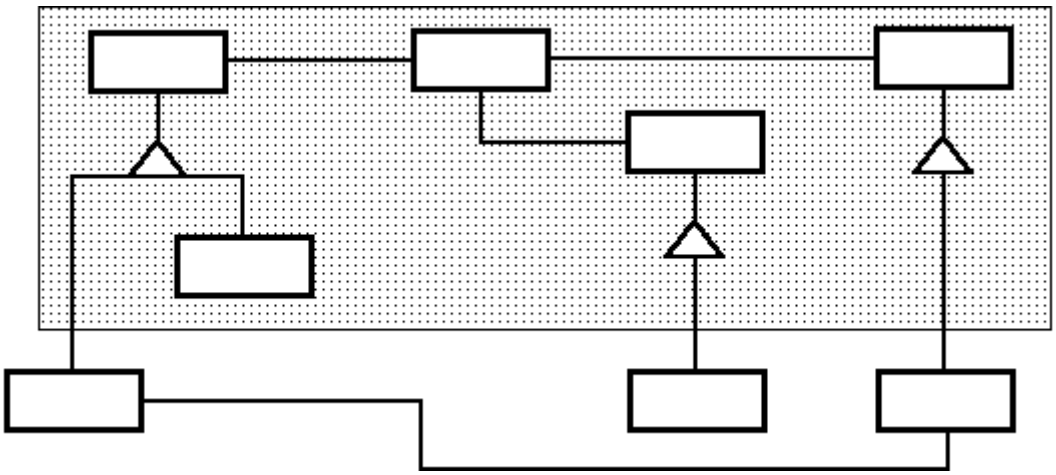
Um *framework* é um esqueleto de implementação de uma aplicação ou de um subsistema de uma aplicação de uma família do mesmo domínio, ou seja, um *framework* é uma estrutura incompleta que pode ser reutilizada para desenvolver aplicações de um domínio particular (SILVA & PRICE, 1998a).

Um *framework* é composto de classes concretas e abstratas e prevê um modelo de interação ou colaboração entre as instâncias das classes definidas por ele. Para que um *framework* seja utilizado no desenvolvimento de aplicações, pode se lançar mão de configuração ou conexão de classes concretas a partir de outras ou derivação de novas classes concretas a partir de classes abstratas.

Os *frameworks* são, portanto, estruturas de classes inter-relacionadas que cooperam entre si, minimizando o esforço de desenvolvimento por parte dos desenvolvedores de aplicações (SILVA & PRICE, 1998b).

Para justificar o desenvolvimento de um *framework*, deve-se identificar no mínimo quatro aplicações do mesmo domínio e, a partir daí, aplicações podem ser desenvolvidas.

A figura 2.1 apresenta a estrutura de um *framework* e uma aplicação criada a partir dele.



Fonte: (SILVA, 2000)

Figura 2.1 - Estrutura de um *framework*

A figura 2.1 apresenta a utilização de um *framework* por parte da aplicação. Os retângulos menores dentro do maior representam as classes inter-relacionadas do *framework*. Os retângulos fora do quadrilátero maior representam as classes da aplicação desenvolvida a partir do *framework*.

Existem dois aspetos fundamentais que caracterizam um *framework*: a) fornecimento da infraestrutura de projeto é disponibilizada ao desenvolvedor a aplicação, minimizando a quantidade de código a ser desenvolvido. Este irá tratar uma aplicação de mesmo domínio, mas com suas especificidades; b) os *frameworks* chamam, não são chamados, ou seja, utilizam o "*Princípio de Hollywood*", pois eles fornecem o fluxo de controle da aplicação (TALIGENT, 1997).

2.2 Características de *Framework*

Um *framework* é desenvolvido para dar suporte a qualquer tipo de aplicação de um mesmo domínio para qual o *framework* foi criado.

Uma característica essencial no desenvolvimento de *frameworks* é a generalidade em relação às funcionalidades do domínio, além de fornecer flexibilidade por meio de características de alterabilidade e extensibilidade (SILVA & PRICE, 1998b).

Frameworks são flexíveis se permitirem alteração na estrutura de suas classes, apoiando, assim, diferentes aplicações de um mesmo domínio. Pela flexibilidade das classes do *framework*, especificidades podem ser acrescentadas a partir de classes do *framework* na aplicação.

2.3 Classificação de *Frameworks*

Frameworks podem ser considerados de três tipos diferentes, dependendo da sua arquitetura de desenvolvimento:

- ***Framework* Caixa-Branca**: Este tipo de *framework* é representado por classes abstratas, que devem ser redefinidas na aplicação pelo mecanismo de herança das classes do *framework*, ou pela sobreposição de métodos *hook* (várias implementações) com recursos da tecnologia orientada a objetos (JOHNSON, 1997).

- ***Framework* Caixa-Preta:** Este tipo de *framework* é definido como conjunto de classes concretas, ou seja, utiliza composição de objetos e delegação ao invés de herança para desenvolver uma nova aplicação, pois não é necessário entender as classes detalhadamente como no outro tipo de *framework.* Toda estrutura necessária para reutilizar as classes como elas estão já foram implementadas no *framework*. Ao contrário do tipo caixa-branca, este tipo de *framework* não permite adaptação das suas especificidades nas aplicações desenvolvidas a partir do *framework* (JOHNSON, 1997).

- ***Framework* Caixa-Cinza:** Este tipo de *framework* é representado por classes concretas e classes abstratas, isto é, possui classes implementadas que podem ser reutilizadas sem acrescentar funcionalidades, e classes que permitem incluir novas funcionalidades de acordo com a necessidade da aplicação (JOHNSON, 1997).

2.4 Vantagens no Uso de *Frameworks*

Embora o desenvolvimento de *frameworks* não seja tarefa fácil, tornou-se meta principal nas indústrias de desenvolvimento de *software* pelas vantagens que isto proporciona:

- Reuso de projeto;
- Reuso de código já implementado;
- Flexibilidade nas classes abstratas permitindo adaptabilidade de funcionalidades nas aplicações desenvolvidas a partir do *framework*.
- Aumento da produtividade no desenvolvimento de aplicações devido ao reuso;
- Menor esforço no desenvolvimento das aplicações em função de parte do código estar pronta;

Essa abordagem possui vantagens como as descritas anteriormente, mas deve-se verificar a real necessidade de desenvolver *frameworks* devido à complexidade dessa tarefa.

A desvantagem desse projeto é a difícil compreensão na utilização de um *framework,* pois aprender a usá-lo requer um esforço considerável, dependendo da complexidade do *framework*, cujo uso poderia levar mais tempo para aprender, do que construir um *software* comum.

Uma forma de entender como o *framework* funciona, é identificar as classes abstratas, e as classes concretas, além de identificar os *Hot Spots*, ou seja, pontos flexíveis do *framework*. Com isso, é possível entender o que os métodos fazem (DAMIAN, 1998).

2.5 *Framework* OCEAN

O *framework* OCEAN foi desenvolvido com a finalidade de suportar a produção de ambientes de desenvolvimento de *software*. OCEAN permite o desenvolvimento de ambientes que manipulem diferentes estruturas de especificação e diferentes funcionalidades (SILVA, 2000).

O *framework* OCEAN é considerado um *framework* tipo caixa cinza, isto é, possui classes implementadas, que podem ser estendidas para outras aplicações e classes abstratas, as quais não possuem implementação e das quais podem ser herdadas características da superclasse incluindo funcionalidades específicas (SILVA, 2000).

Um ambiente de desenvolvimento de *software* específico pode ser construído a partir de uma subclasse concreta de *EnvironmentManager* (classe abstrata do OCEAN). A partir dessa classe, são definidas as características de um ambiente específico:

- O tipo de especificação a ser tratado pelo ambiente (cada especificação define um conjunto de tipos de modelos e cada modelo define um conjunto de tipos de conceitos);
- Mecanismo de visualização e edição associado a cada modelo e conceito tratado no ambiente específico;
- Mecanismo de armazenamento de especificações no ambiente;
- Ferramentas para manipular especificações.

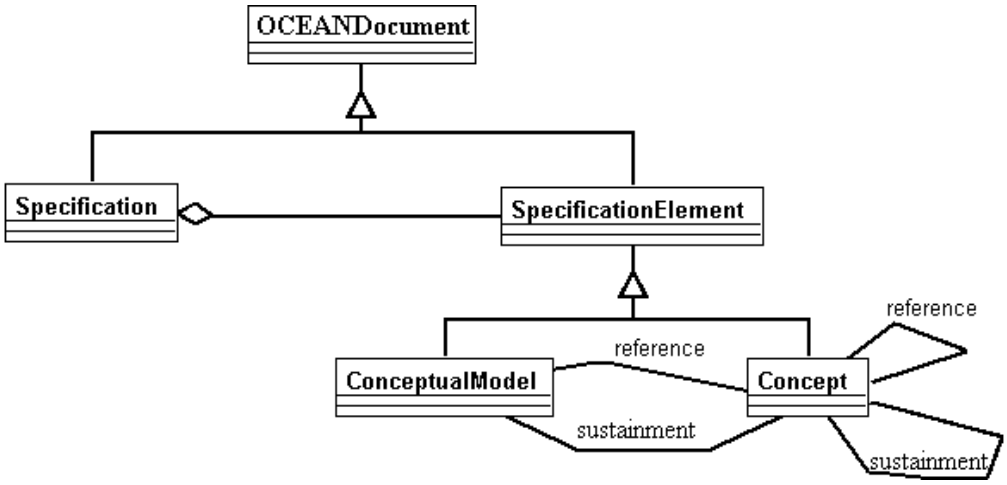
Devido a sua flexibilidade, o OCEAN permite produção e alteração de especificações. Oferece, também, alicerce à criação de estruturas de documentos, edição semântica de especificações e composição de ações de edição complexas por meio de ferramentas (SILVA, 2000).

2.5.1 Suporte à Criação de Documentos a partir do *Framework* OCEAN

O OCEAN suporta a definição de diferentes tipos de estruturas de documentos. Por exemplo, no ambiente SEA são manipulados documentos de estruturas distintas, tais como: *Cookbook* Ativo, Especificação Estrutural de Interfaces de Componentes, Especificação OO (orientada a objetos), Projeto de *Workflow*.

2.5.1.1 Estrutura de Documentos a partir do OCEAN

A criação de estruturas de documentos é realizada por meio da herança das classes abstratas suportadas no *framework* OCEAN. A figura 2.2 representa a hierarquia das classes que definem a estrutura de documentos a partir do *framework* OCEAN.



Fonte: (SILVA, 2000)

Figura 2.2 - Classes de uma estrutura de um documento a partir do *Framework* OCEAN

Na estrutura apresentada pela figura 2.2, uma instância de *Specification* (uma especificação) agrega elementos de especificação, isto é, agrega as instâncias de subclasses de *Concept* e *ConceptualModel*. A classe *Concept* do OCEAN permite a criação de todos os conceitos tratados no ambiente, por meio de subclasses concretas que modelam cada tipo de conceito definido no ambiente. A classe *ConceptualModel* permite a criação dos tipos de modelos do ambiente por suas subclasses concretas que descrevem os modelos tratados no ambiente. Os modelos criados a partir da classe *ConceptualModel* referenciam os conceitos definidos no ambiente.

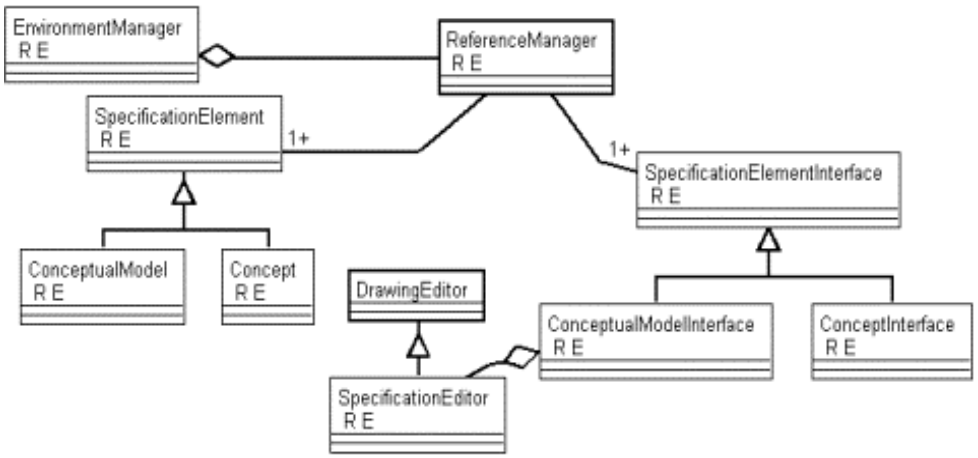
2.5.2 Visualização de Elementos de Especificação

O OCEAN dispõe de uma classe gerenciadora, *ReferenceManager*, que associa cada elemento de especificação a um mecanismo de visualização. Para essa representação visual, o

OCEAN criou subclasses de classe do *framework HotDraw* (BRANDT, 1995), pois este *framework* permite criar e editar elementos visuais. Por isso, todos os elementos de especificação criados estão associados ao *HotDraw* (classe *DrawingEditor* da figura 2.2 que pertence ao OCEAN), e, assim, para cada conceito definido deve-se criar uma figura do conceito.

O gerente de referência do OCEAN permite que elementos de especificação e os mecanismos de visualização possam ser desacoplados, permitindo o reuso separado de cada estrutura e a associação entre elas.

A figura 2.3 apresenta a estrutura de edição de elementos de especificação a partir do OCEAN.



Fonte: (SILVA, 2000)

Figura 2.3 - Estrutura de edição de elementos de especificação a partir do OCEAN

Na figura 2.3 apresentada, a classe *SpecificationElementInterface* demonstra a estrutura genérica dos mecanismos de visualização dos elementos de uma especificação. Essa classe contém duas subclasses, *ConceptInterface* responsável pela visualização dos conceitos, e a subclasse *ConceptualModelInterface* responsável pelo mecanismo de visualização de modelos e agrega um editor gráfico.

A partir disso, pode-se definir classes concretas de *SpecificationElementInterface*, criando novos mecanismos de visualização de modelos e conceitos.

2.5.3 Interligação de Elementos de Especificação do *Framework* OCEAN

Na estrutura de especificações definida no *framework* OCEAN são adotados mecanismos que fornecem ligação entre os elementos pertencentes à especificação, modelos e conceitos definidos em cada especificação.

O OCEAN oferece três maneiras para definir ligações entre os elementos de especificação. Estes mecanismos de ligação são utilizados pelo ambiente de desenvolvimento de *software*, SEA. Pode-se estabelecer ligações entre os elementos de uma especificação através de *links*, associações de sustentação e associações de referência.

2.5.2.1 Mecanismo de ligação por links

Link é um conceito que pode ser associado a qualquer elemento de especificação. A associação de *links* a elementos de especificação é utilizada para apontar outros documentos que podem ser uma especificação ou um elemento de uma especificação em si.

A associação de *links* possui duas finalidades nos ambientes criados a partir do OCEAN: disponibiliza a criação de rotas de acesso às especificações criadas e estabelece ligações semânticas entre os elementos de especificação. A utilização de *links* com ligação semântica é definida na estrutura semântica de uma especificação.

Os *links* semânticos possibilitam estabelecer ligação semântica entre os elementos de uma especificação, não influenciando diretamente a estrutura do elemento que possui o *link* e do elemento apontado pelo *link*, ou seja, a remoção de um dos elementos de especificação não afeta a alteração da estrutura do outro elemento, apenas pode acontecer a incompletude de uma especificação.

2.5.2.2 Mecanismo de ligação por sustentação

Uma associação de sustentação é outro tipo de recurso previsto no *framework* OCEAN para ligação semântica entre dois elementos de especificações diferentes, no qual um elemento é definido como sustentado e o outro, sustentador.

Nesse mecanismo de ligação entre elementos de especificação, a semântica dessa associação define que a existência de um elemento sustentado de uma especificação só depende da existência do elemento sustentador. Assim, por exemplo, a existência de atributos e métodos pertencentes a uma classe, só existem enquanto a classe existir.

2.5.2.3 Mecanismo de ligação por referência

Nesse tipo de ligação entre elementos de especificação, associação por referência, foi definido que parte da definição de um determinado elemento de especificação, ou seja, o elemento referenciador, depende da referência a outro elemento, o referenciado. Caso, o elemento referenciado seja removido, o elemento que o referencia não o será, porém, sua estrutura ficará incompleta, comprometendo, assim, a consistência das informações.

2.6 Flexibilidade do *Framework* OCEAN

O *framework* OCEAN não se restringe a fornecer suporte a diferentes estruturas de especificação, pois também permite definir funcionalidades para atuar sobre as especificações dos ambientes desenvolvidos a partir dele. A seguir, são apresentadas algumas funcionalidades disponíveis no OCEAN.

- **Funcionalidades supridas pelo** OCEAN: São funcionalidades gerais, aplicadas a distintos ambientes, estabelecidos no *framework* para serem reutilizadas. Parte destas funcionalidades são totalmente definidas no *framework*, ou seja, são totalmente reusadas pelo ambiente criado a partir do OCEAN e parte das funcionalidades precisam ser acrescentadas em subclasses concretas das classes que são definidas como classes redefiníveis do *framework*;

- **Funcionalidades supridas pelos editores de modelos**: Os editores de modelos são responsáveis pela criação, modificação e remoção de conceitos adicionados aos modelos de uma especificação. Essas características são permitidas em funcionalidades de edição presentes no OCEAN;

- **Funcionalidades supridas por ferramentas**: As ferramentas possuem a finalidade de permitir ações de edição, análise ou transformação sobre conceitos, modelos ou especificações.

A inclusão de ferramentas em um ambiente não exige modificação no resto de sua estrutura. Por isso, há flexibilidade em relação às possibilidades de criação de ferramentas oferecidas pelo OCEAN.

O *framework* fornece uma definição geral de ferramenta que pode ser particularizada na definição de ferramentas específicas. As ferramentas sob o *framework* OCEAN podem ser classificadas como:

- **Ferramentas de edição:** têm a finalidade de ler e modificar uma especificação. Nessa classificação estão inseridos os editores de modelos;

- **Ferramentas de análise:** Possuem a finalidade de ler uma determinada especificação, mas não de alterá-la. Estruturas de especificação disponíveis no *framework* OCEAN possuem definição semântica e, por isso, é possível criar ferramentas de verificação de consistência, de avaliação de qualidade da especificação, por exemplo. Elas produzem descrições de características de uma especificação;

- **Ferramentas de transformação:** Esta classificação de ferramenta realiza a ligação de especificações do ambiente a elementos externos. Possuem a capacidade de importar ou exportar especificações.

2.7 Conclusão

O trabalho desenvolvido é uma extensão de um ambiente de desenvolvimento de *software* criado a partir do *framework* OCEAN e por isso, devido à proposta de trabalho, a descrição sob o *Framework* OCEAN é importante para entender a forma de como o trabalho foi desenvolvido, pois todas as subclasses que serão criadas, são subclasses de classes do *framework*.

O ambiente de desenvolvimento de *software* que será descrito no próximo capítulo, possui algumas deficiências e as novas funcionalidades adicionadas a esse ambiente tentam suprir tal carência. Maiores detalhes estarão no próximo capítulo.

O próximo capítulo enfatiza o ambiente SEA, desenvolvido a partir do *framework* OCEAN e sua extensão, suporte de *workflow* a esse ambiente.

3 *Workflow* e Requisitos CMMI em um Ambiente de Desenvolvimento de *Software* - SEA

Este capítulo apresenta o ambiente de desenvolvimento de *software* SEA (SILVA, 2000) que oferece suporte ao desenvolvimento de aplicações, componentes e *frameworks*, usando *Workflow* como gerenciador das atividades dos processos com o uso de alguns requisitos do nível 2 do *Capability Maturity Model Integration* (CMMI).

Assim como *framework* OCEAN, o ambiente SEA foi desenvolvido na linguagem de programação orientada a objetos*, Smalltalk* utilizando a ferramenta *Visual Works*.

A descrição do ambiente de desenvolvimento de *software* SEA é necessária, pois, a presente abordagem acrescenta novas particularidades a esse ambiente, bem como a sua extensão desenvolvida por (SCHEIDT, 2002), a qual fornece suporte ao gerenciamento das atividades.

Como o ambiente SEA é parte do desenvolvimento da tese de doutorado de (SILVA, 2000) – "*Suporte ao desenvolvimento e uso de frameworks e componentes"* e a extensão do ambiente SEA é a dissertação de mestrado de (SCHEIDT, 2002) – "Estudo e aplicação de *workflow* e requisitos de CMM em um ambiente de desenvolvimento de *software*", ambas são referências deste capítulo.

3.1 Ambiente de Desenvolvimento de *Software* – SEA

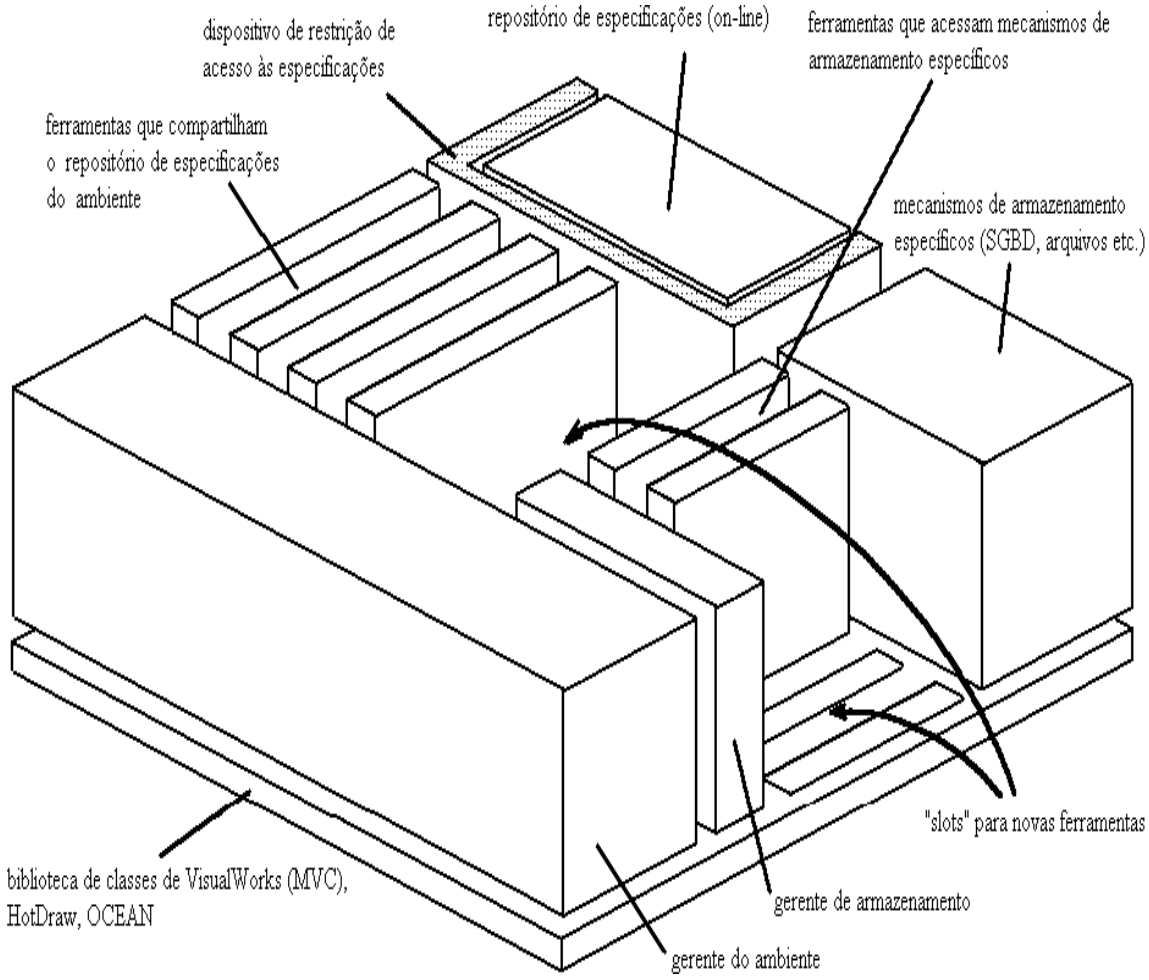
O ambiente SEA foi implementado a partir do *framework* OCEAN com a finalidade de desenvolvimento e uso de artefatos reutilizáveis. SEA fornece suporte ao desenvolvimento de *software* e dispõe também da utilização integrada das abordagens de desenvolvimento orientado a componentes.

SEA prevê o desenvolvimento de aplicações baseadas no paradigma de orientação a objeto, *frameworks* e componentes, e consiste em desenvolver a especificação de projeto

desses artefatos, utilizando as técnicas de *Unified Modeling Language* (*OBJECT MANAGEMENT GROUP*, 2004).

SEA permite, ainda, a ligação semântica entre o artefato de *software* e o *framework* que o originou através de um elo denominado *link*.

A figura 3.1 apresenta a estrutura do ambiente SEA. Essa estrutura é reutilizada para qualquer ambiente de desenvolvimento de *software* construído a partir do OCEAN.



Fonte: (SILVA, 2000)

Figura 3.1 - Estrutura do ambiente SEA constituído a partir do *framework* OCEAN.

Na figura 3.1, observa-se que ambiente SEA constitui-se de um repositório de especificações utilizado por diferentes ferramentas. O gerente do ambiente constitui a interface com usuário e permite a manipulação das especificações pertencentes ao repositório

do ambiente por meio das ferramentas. Através de um gerente de armazenamento, especificações pertencentes ao repositório de especificações do ambiente podem ser gravadas em dispositivos de armazenamento e especificações pertencentes a esses dispositivos podem ser carregadas no compartimento de conceitos.

A figura 3.2 apresenta a interface do ambiente SEA. Através da janela da figura 3.2 é possível criar uma estrutura de especificação: Estrutura de Classes, Interface de Componentes, *CookBook* Ativo, Projeto de *Workflow*, Estrutura de Documentos Textuais, e ferramentas de consistência.

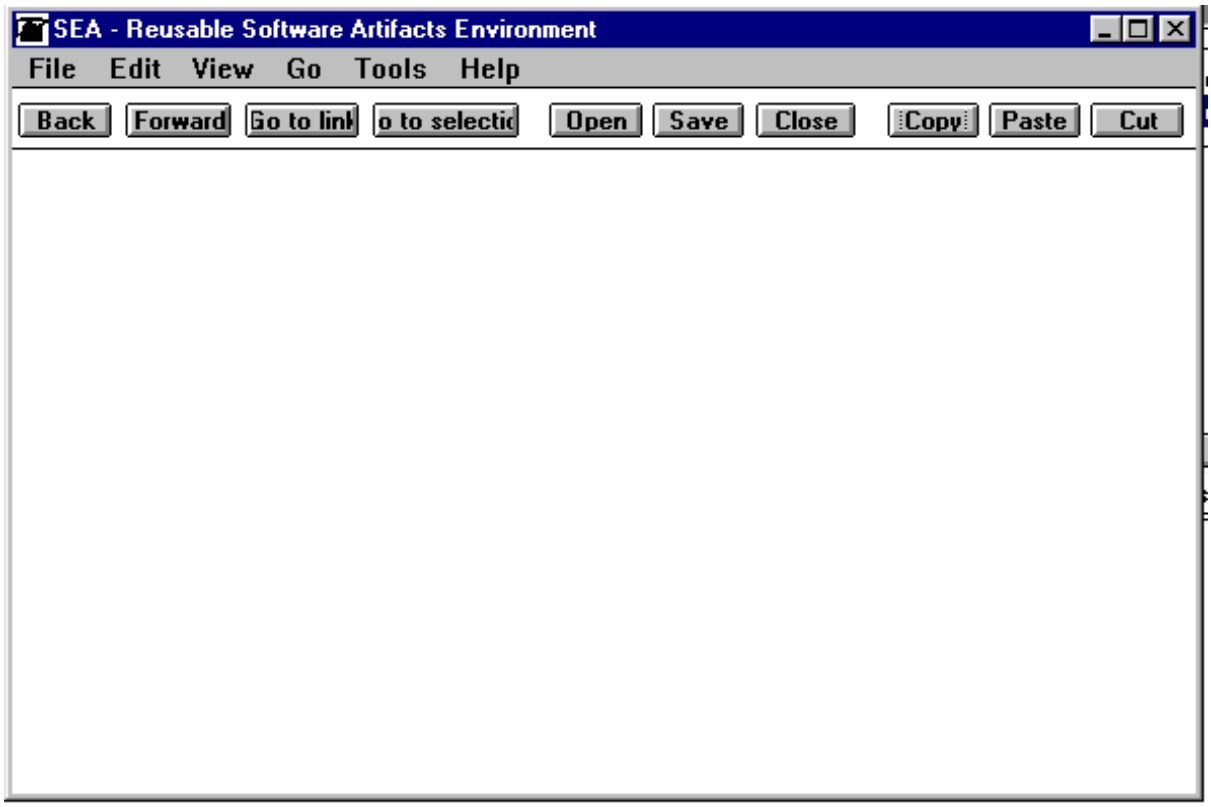


Figura 3.2 – Interface Inicial do Ambiente SEA

3.1.1 Estrutura Orientada a Objetos no Ambiente SEA

O ambiente SEA em sua primeira versão permitia três tipos de especificação (SILVA, 2000):

- Especificação Orientada a Objetos (OO), que permite descrever *framework*, uma aplicação, componente ou um componente flexível;
- Especificação de interface de componentes e;
- *Cookbook* ativo que serve para auxiliar no uso de um *framework* previamente desenvolvido.

Os outros tipos de estrutura de especificação introduzidos ao ambiente, Projeto de *Workflow* por (SCHEIDT, 2002) e Estrutura de Documentos Textuais proposta deste trabalho, também fazem parte das especificações disponíveis no SEA, sendo que ambas foram inseridas ao ambiente como contribuição de trabalhos posteriores.

A figura 3.3 apresenta os tipos de especificação disponíveis no ambiente SEA. Cada estrutura de especificação possui seus modelos e conceitos específicos.

Figura 3.3 - Tipos de Especificações oferecidos pelo Ambiente SEA

3.1.1.1 Modelagem para Especificação OO no ambiente SEA

O ambiente SEA utiliza técnicas de *Unified Modeling Language* (*OBJECT MANAGEMENT GROUP*, 2004) para produzir especificações OO e uma técnica adicional não prevista em UML que possui a finalidade de descrever os algoritmos dos métodos. Uma

especificação OO pode ser um projeto de um *framework*, de uma aplicação, de um componente ou de um componente flexível.

O ambiente SEA fornece as seguintes técnicas para modelagem:

- Diagrama de casos de uso;
- Diagrama de atividades;
- Diagrama de classes;
- Diagrama de seqüência;
- Diagrama de transição de estados;
- Diagrama de corpo de método.

Extensões foram adicionadas ao ambiente. Essas extensões são definidas para:

- Representar conceitos do domínio de *frameworks*, não fornecidos pelas técnicas de UML;
- Anexar características de UML, como a possibilidade de definir restrições na ordem dos casos de uso e a associação semântica aos estados;
- Permitir a descrição dos algoritmos dos métodos na especificação de projeto;
- Expressar a ligação semântica entre os elementos de uma especificação;
- Permitir que especificações sejam tratadas como hiperdocumentos, de forma que os elementos de uma delas tenham *links* interligados que apontem a outros documentos, ou seja, especificações ou elementos de especificação.

3.1.1.2 Desenvolvimento e uso de Componentes no Ambiente SEA

Componentes no ambiente SEA são tratados pelas especificações OO, semelhante ao que ocorre no desenvolvimento de *frameworks* e aplicações.

A especificação de um componente é caracterizada por parte da especificação correspondente à descrição de uma interface, isto é, uma estrutura de classes responsável pela comunicação do componente com o meio externo e que possui uma especificação individual que define sua estrutura e sua dinâmica comportamental.

Uma estrutura de classes de um componente reutiliza uma interface, eleita em uma biblioteca de interface de componentes.

A especificação estrutural de uma interface de componente interliga os métodos fornecidos, os métodos requeridos e a associação destes métodos a cada canal definido na interface.

A questão da descrição comportamental da interface de um componente é definida para verificar se há restrições associadas à ordem em que os métodos estão sendo invocados. A falta de restrições indica que qualquer método fornecido ou requerido pode ser invocado a qualquer momento da vida de um componente.

No ambiente SEA foi adotada uma Rede de *Petri*, que é constituída de lugares, transições, arcos que interligam lugares e transições e uma marcação inicial definida por uma quantidade de fichas em cada lugar da rede.

Na Rede de *Petri* é necessário associar cada par (canal, método) definido na estrutura da interface a uma ou mais transições.

3.1.1.3 Uso de Cookbook Ativo para a construção de especificações sob um framework no ambiente SEA

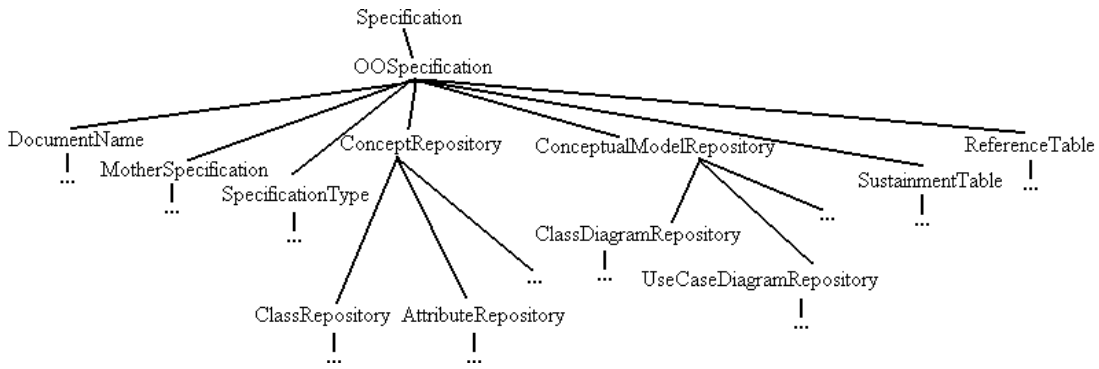
Um *cookbook* ativo tem a finalidade de orientar na utilização de *frameworks* por meio de documentos manuseáveis.

As estruturas de especificações presentes no ambiente SEA para fornecer suporte ao desenvolvimento de especificações de *frameworks* e de *cookbooks* ativos incluem duas abordagens: a) possibilitam verificar as características de *frameworks* a serem considerados no desenvolvimento de um *cookbook* ativo; b) incluem a capacidade de avaliar a qualidade do *cookbook* ativo produzido para um *framework*.

3.1.2 Estrutura Semântica de especificação OO no ambiente SEA

Como mencionado anteriormente, uma especificação OO é um dos tipos de especificação fornecidos pelo ambiente SEA com a finalidade de produzir especificações de projeto de *frameworks*, aplicações ou componentes, em que uma estrutura de uma especificação é caracterizada a partir de um conjunto de tipos de modelo, de um conjunto de tipos de conceito e de um conjunto de associações entre os elementos destes conjuntos.

A figura 3.4 apresenta a estrutura semântica de uma especificação OO no ambiente SEA.



Fonte: (SILVA, 2000)

Figura 3.4 - Parte da árvore de derivação de uma especificação OO

3.2 *Workflow* e CMMI no Ambiente SEA

Devido à flexibilidade do *framework* OCEAN em possuir uma estrutura extensível, permitindo que novas funcionalidades sejam adaptadas a qualquer ambiente desenvolvido a partir do OCEAN, foi possível introduzir ao ambiente SEA novas propriedades (SCHEIDT, 2002).

O ambiente SEA fornece toda uma estrutura em relação ao aspecto tecnológico do desenvolvimento de *software*, mas não possuía a capacidade de criar e controlar o fluxo das

atividades que estão sendo realizadas durante o desenvolvimento de um *software*, permitindo a construção de especificações, verificação de consistência e geração de código.

Para suprir a ausência de gerenciamento de processos foram adaptadas ao ambiente SEA, técnicas de *Workflow* e adotados conceitos de CMMI ao ambiente para suportar gerenciamento na realização das atividades com a finalidade de prover qualidade ao desenvolvimento de *software* (SCHEIDT, 2002).

Define-se um sistema de *workflow* como sendo um conjunto de atividades organizadas com a finalidade de realizar um processo de negócio (*WORKFLOW MANAGEMENT COALITION*, 1999). Um processo de negócio pode ser compreendido como um conjunto de uma ou mais atividades relacionadas, que coletivamente atingem um objetivo de negócios, dentro do contexto de uma estrutura organizacional. As atividades podem ser humanas, ou automatizadas (ZSCHORNACK, 2003).

Partindo dessa deficiência foi criado um editor de *workflow* no ambiente SEA que, entre outras aptidões, permitiu construir o plano de desenvolvimento do *software*, atribuindo tarefas aos atores envolvidos nos processos e ordenando a execução dos processos em um projeto. Com isso, foi acrescentado ao ambiente SEA, mais um tipo de especificação, o "Projeto de *Workflow"* (SCHEIDT, 2002).

Um Sistema de Gerenciamento de *Workflow* permite a definição, o gerenciamento e a execução de *workflows* num processo de negócio (*WORKFLOW MANAGEMENT COALITION*, 1999). Partindo disso, foi criado um sistema gerenciador de *workflow* para inserir qualidade no desenvolvimento dos projetos, que permite acompanhar e supervisionar a execução dos processos no desenvolvimento de *software* e adaptou-se a esse sistema gerenciador, alguns requisitos de CMMI (PAs Planejamento de Projeto, Monitorização e Controle de Projeto e Garantia de Qualidade do Processo e Produto) à sua estrutura para atingir as metas definidas nessas PAs. Pode-se concluir, no entanto, que o novo ambiente suporta apenas parte do nível 2 de CMMI.

A ferramenta de *Workflow* inserida ao ambiente SEA trata integralmente as PAs Planejamento de Projeto e Monitorização e Controle de Projeto e, parcialmente, a PA

Garantia de Qualidade do Processo e Produto, porém, outros requisitos de CMMI podem ser relacionados às atividades de *Workflow (SCHEIDT, 2002)*.

As novas funcionalidades inseridas ao ambiente SEA são ferramentas de suporte a *workflow*, gerenciamento de processos, planejamento e controle dos projetos. As ferramentas de gerenciamento são auxiliadas por requisitos do modelo de CMMI e, por isso, pode-se dizer que o ambiente SEA com as novas funcionalidades, compreende parte do nível 2 de CMMI, ou seja, suporta a capacidade de gerenciamento.

A seguir, são descritos o editor de *workflow* e o sistema gerenciador de *workflow* no ambiente SEA. Isto se faz necessário, pois o trabalho irá tentar suprir uma carência encontrada no ambiente SEA mesmo após a inclusão da nova especificação.

3.2.1 Modelagem do Projeto de *Workflow*

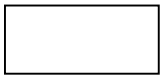
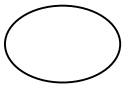
As técnicas existentes de modelagem de *workflow* devem ter, como princípio básico, a diminuição dos problemas de controle do trabalho nos processos de negócios. Essas técnicas apóiam-se na dinamicidade do processo e necessitam oferecer recursos para representar o fluxo de trabalho ao longo do processo de modelagem (HOLLINGSWORTH, 1995).

A técnica de (SCHEIDT, 2002) utilizou a modelagem de gatilhos com algumas inclusões, porque atende aos requisitos do projeto (notação que permite representar primitivas de sincronismo, dependência entre as atividades, forma de disparo das atividades, pré-condições para a ocorrência de uma atividade) e porque sua visualização é simples de compreender.

De acordo com a modelagem utilizada por (SCHEIDT, 2002), pode-se observar a terminologia adotada conforme demonstrada na tabela 3.1, onde, cada coluna de um processo modelado representa um participante do *workflow* e, a partir destas colunas, pode-se identificar as *worklists* de cada participante.

A técnica de gatilhos adota a seguinte terminologia, conforme apresenta a tabela 3.1.

Tabela 3.1 - Terminologia da técnica de modelagem de *workflow* gatilhos com inclusões

SÍMBOLO	DESCRIÇÃO
	Representa Início e Fim de Processo
	Representa uma Atividade
	Representa uma Tomada de Decisão
	Representa um Disparo de Atividade
	Representa uma Sincronização das Atividades
S(N)	S – seta (n) – dígito indicando o número da seta
	Cada Coluna contém um Participante e a Sua Lista de Trabalhos

Fonte: (SCHEIDT, 2002)

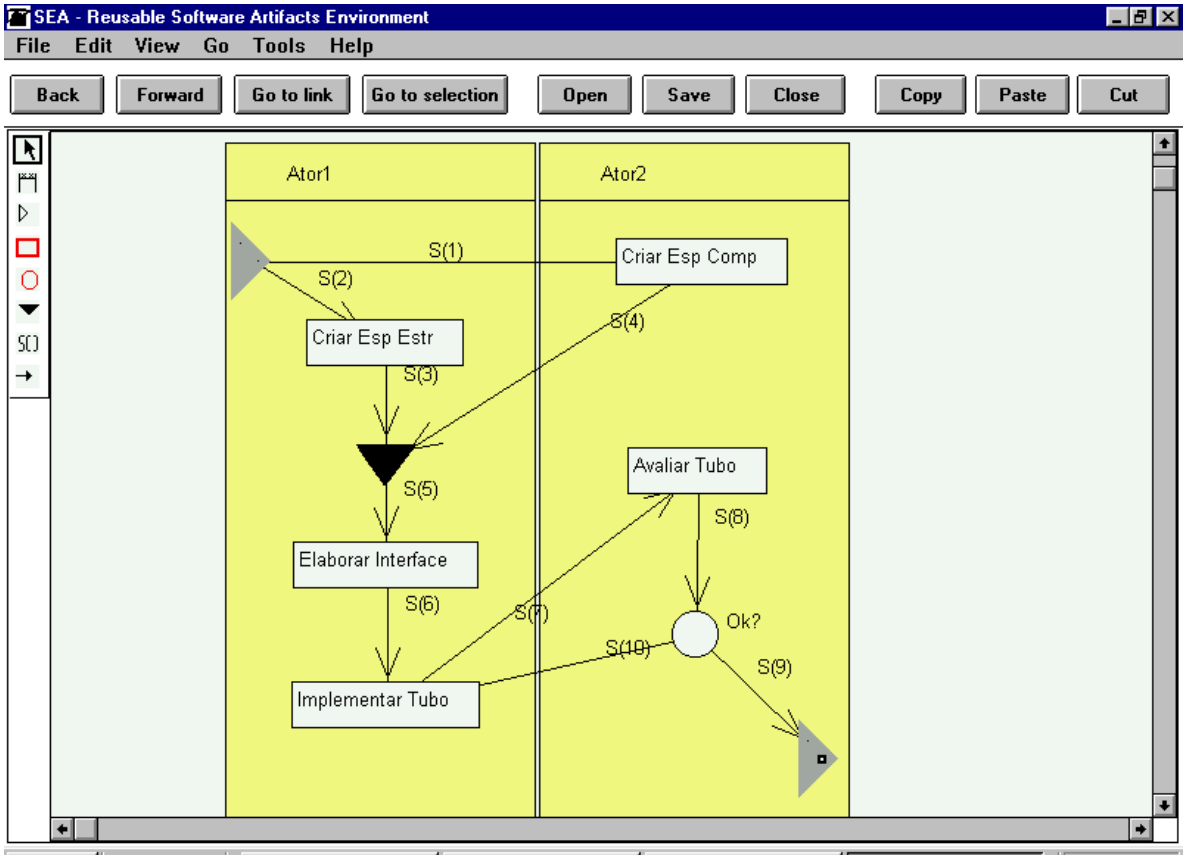
Para modelar um sistema de *workflow,* deve-se seguir algumas prioridades: determinar o sistema de *workflow*; estabelecer os participantes; identificar as atividades executadas sob a responsabilidade de cada participante; identificar como cada atividade é disparada, e gerar o modelo.

3.2.2 Editor de *Workflow* no SEA

O editor de *workflow* permite que sejam editados e manipulados conceitos como, *Activity*, *Actor*, *Dart*, *Decision*, *Trigger*, *Beginend* e *Synchronize*.

Uma atividade é um conjunto de eventos que acontece sob a responsabilidade de um ator. Um ator é responsável pela execução de tais atividades. Com esses conceitos, a modelagem de desenvolvimento de *software* pode ser elaborada.

Na Figura 3.5 é representado um modelo de um diagrama de *workflow* modelado no próprio editor.



Fonte: (SCHEIDT, 2002)

Figura 3.5 - Editor de *Workflow*

Normalmente, uma atividade dispara outra que pode ser executada por outro participante ou não. Uma atividade pode ser acionada paralelamente a outra atividade. Setas servem para indicar a ordem das atividades que, dependendo da decisão a ser tomada, alteram a ordem dos processos para serem executadas. Uma decisão tem um condicionador de ocorrência do tipo "*if*".

O número de conceitos em diagrama de *workflow* varia, dependendo de cada caso. Porém, o único que tem seu número delimitado é *BeginEnd*, pois um *workflow* só tem um início e um fim (SCHEIDT, 2002).

O editor de *workflow* possui algumas restrições que podem ser encontradas com detalhes em (SCHEIDT, 2002).

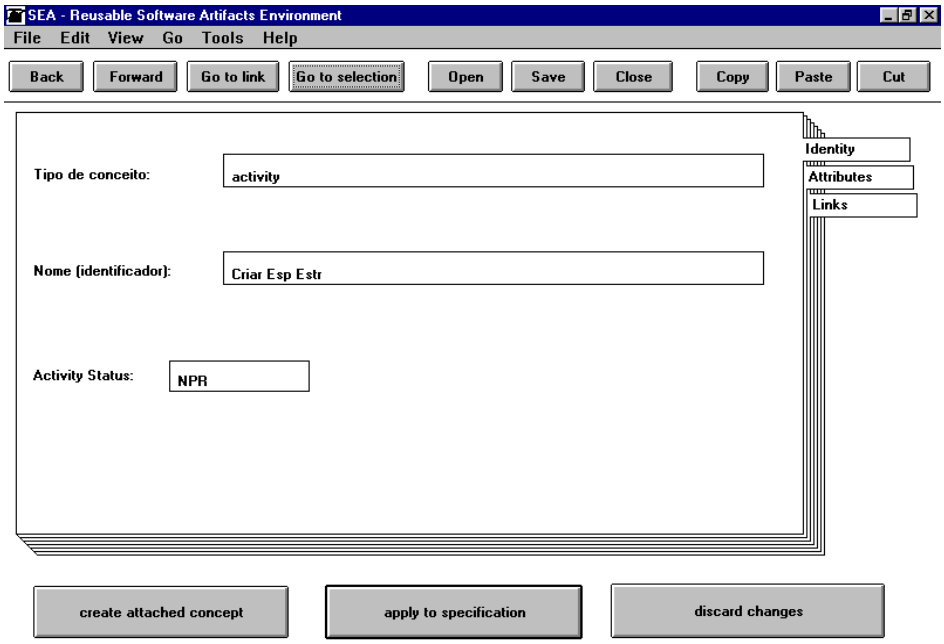
3.2.3 Janelas de edição dos conceitos de *workflow*

Cada conceito de *workflow* possui suas particularidades, portanto, cada conceito possui sua janela de edição específica, e possui propriedades específicas que podem ser alteradas para garantir a consistência de cada um dos conceitos.

Uma atividade possui uma janela de edição, dividida em três folhas: *Identity*, *Attributes* e *Links.*

A folha *Identity* possui o tipo de conceito que está sendo tratado (*activity*), o nome da atividade (nome estabelecido ao ser planejado o *workflow*) e o estado corrente da atividade (que pode ser atividade processada, atividade não processada, atividade em andamento, atividade irregular). A folha *Attributes* contém os atributos referentes a uma atividade, dentre os quais**,** alguns são obrigatórios: tipo de atividade (humana ou automatizada), descrição da atividade, data prevista para o início e para o término da mesma, o esforço a ser medido (cálculo sobre o tempo e esforço que foram gastos para processar uma atividade) pertencente a um trabalho futuro desse ambiente. A folha *Links* permite a edição do *link* a ser criado e a escolha da especificação/documento ao qual a atividade está relacionada. O *link* é obrigatório em qualquer atividade considerada automatizada. O *link* possibilita a ligação entre *workflow*, o ambiente de desenvolvimento de *software* e o gerenciador de *workflow*.

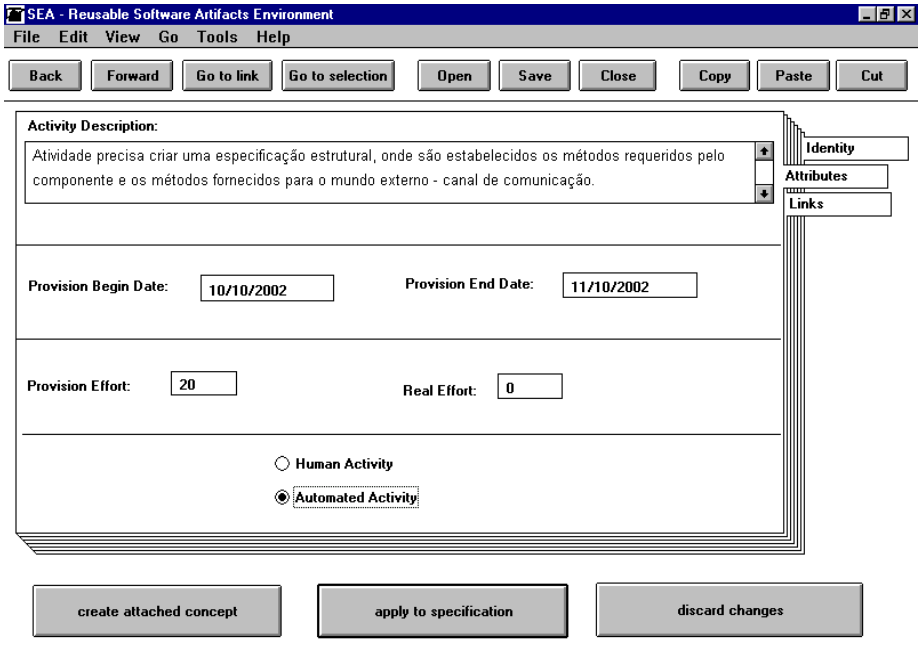
A figura 3.6 apresenta a folha de edição *Identify* pertencente a uma atividade a ser realizada.



Fonte: (SCHEIDT, 2002)

Figura 3.6 - Folha de edição de *Identify* de uma atividade

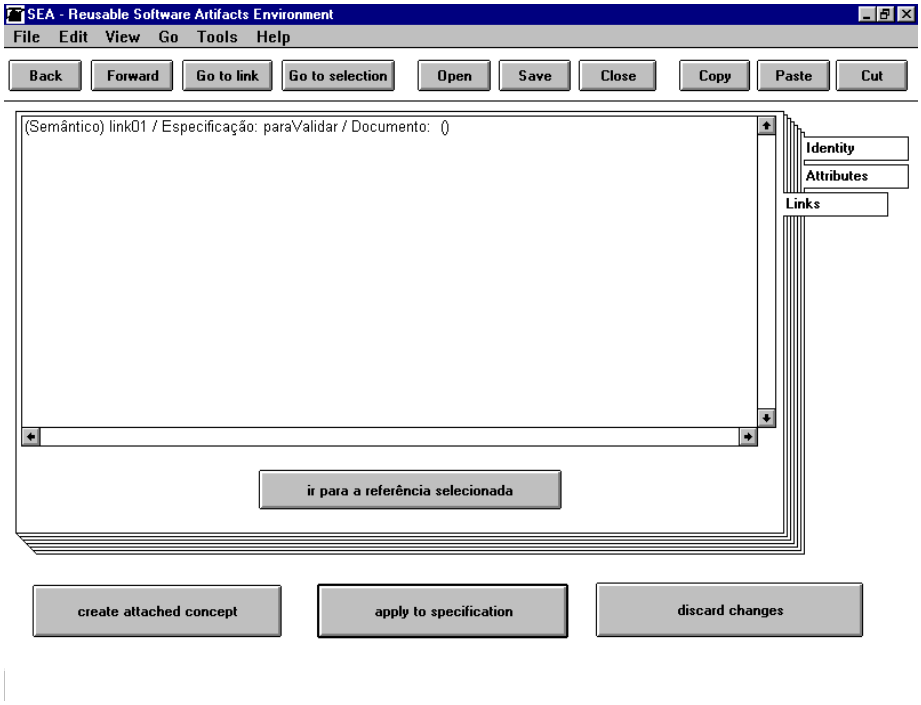
A figura 3.7 apresenta a folha de edição *Attributes* de uma atividade a ser realizada.



Fonte: (SCHEIDT, 2002)

Figura 3.7 - Folha de edição de *Attributes* de uma atividade

A figura 3.8 apresenta a folha de edição *Links* de uma atividade a ser realizada.



Fonte: (SCHEIDT, 2002)

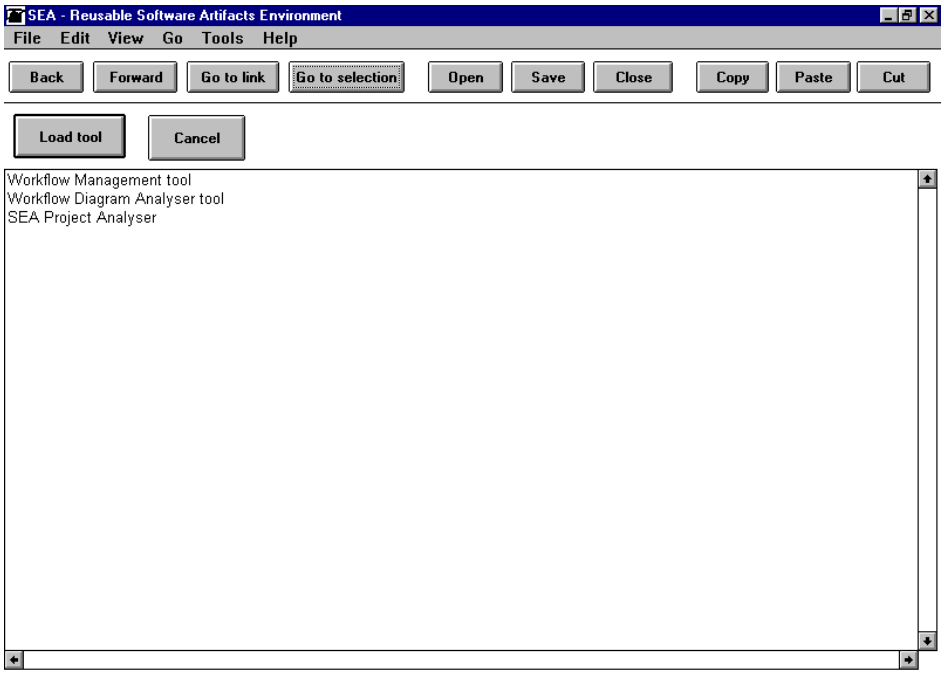
Figura 3.8 - Folha de edição *Links* de uma atividade

3.2.4 Verificação de Consistência da Especificação Projeto de *Workflow*

Dentre alguns recursos do ambiente SEA com a especificação Projeto de *Workflow*, contribuição de (SCHEIDT, 2002), para testar a consistência de uma especificação, existe um recurso denominado ferramenta de análise, em que há três tipos que analisam a solidez das informações presentes no Projeto de *Workflow*.

O ambiente dispõe de uma ferramenta de análise para o modelo, chamada *WorkflowDiagramAnalyserTool* e uma ferramenta de análise para a especificação, denominada *ProjectAnalyser*.

Caso seja escolhido o analisador de modelos, o modelo descrito do diagrama de *workflow* atual é analisado. A figura 3.9, mostra a janela que apresenta os analisadores do ambiente SEA.



Fonte: (SCHEIDT, 2002)

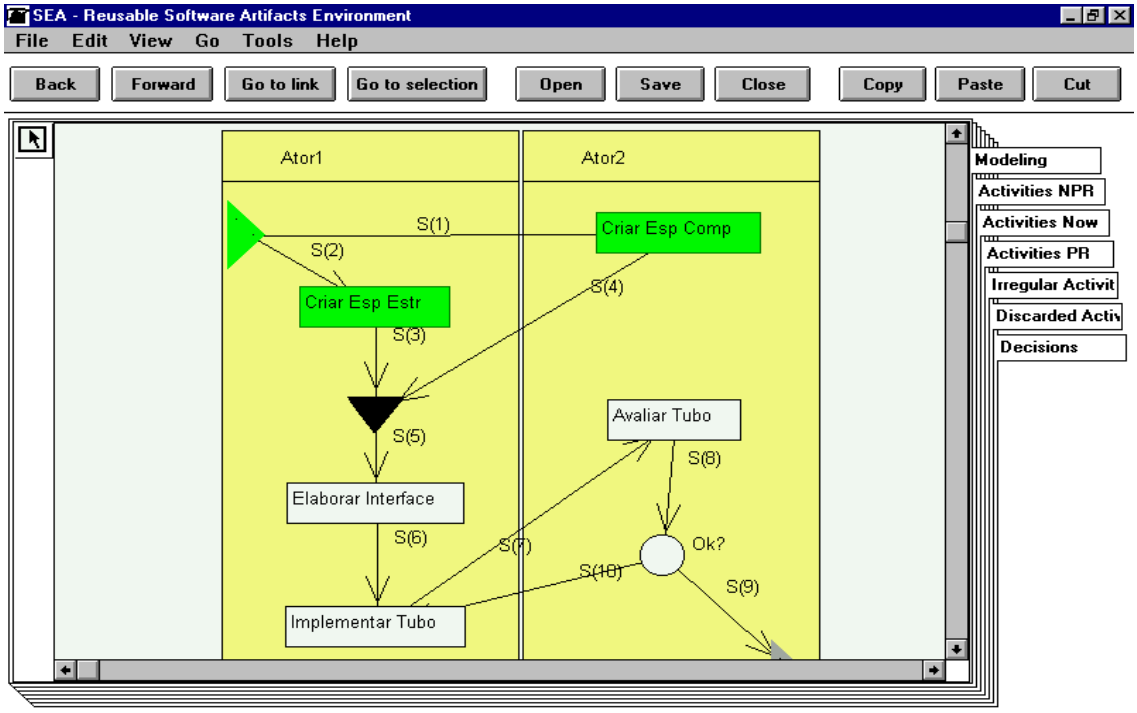
Figura 3.9 - Janela de escolha dos analisadores de *workflow*

Caso seja escolhido o analisador de especificações, ele leva à análise do modelo, pois ainda existe apenas um modelo de *workflow* por especificação. Tanto o analisador de modelos quanto o de especificações levam à mesma ação.

3.2.5 Sistema Gerenciador de *Workflow* no ambiente SEA

O sistema gerenciador de *workflow* é uma ferramenta que fornece suporte ao controle das atividades a serem realizadas. Esta ferramenta é uma estrutura funcional que reutiliza os procedimentos de edição semântica presentes no *framework* OCEAN.

Esta ferramenta, no ambiente SEA chama-se *Workflow Management Tool* e tem como objetivo verificar e gerenciar as atividades e processos a serem realizados. O ambiente permite que essa ferramenta seja acionada caso a especificação de *workflow* esteja "*Frozen*", isto é, congelada. Essa opção só é permitida caso a especificação esteja validada. Assim, apenas especificações de *workflow* consistentes podem ser gerenciadas.



Fonte: (SCHEIDT, 2002)

Figura 3.10 - Gerenciamento do modelo de *workflow* e o andamento dos processos

Na Figura 3.10 é mostrada uma das telas do SGW/SEA, a tela *Modeling*, que demonstra o modelo de *workflow* que está sendo gerenciado e o andamento dos seus processos.

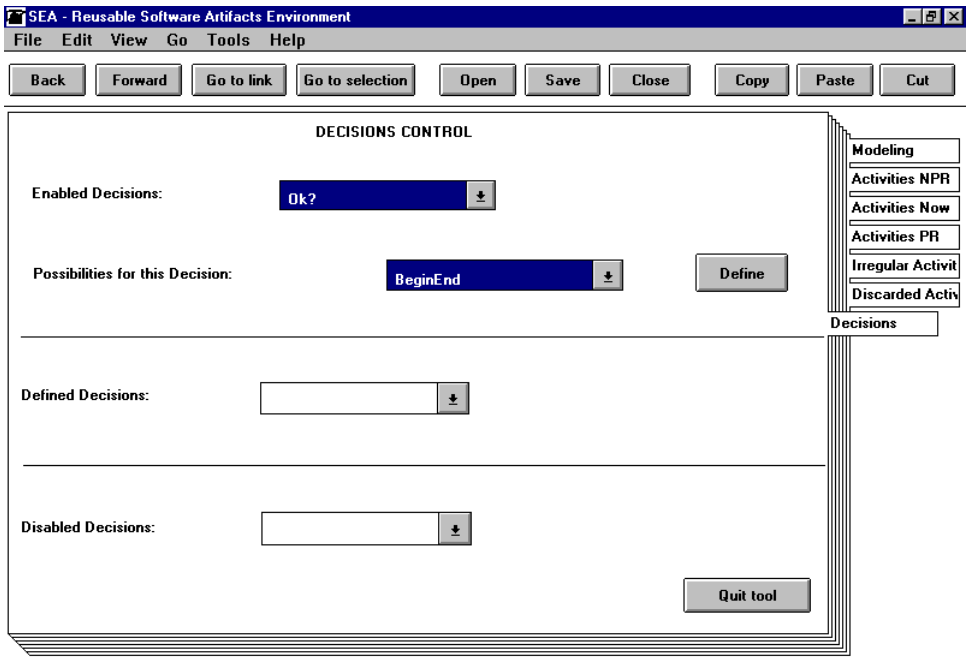
O gerenciador de *workflow* controla os estados das atividades por meio de cores já definidas: a cor branca representa uma atividade que ainda não foi processada; a azul, uma atividade em andamento; a cor verde, uma atividade já processada e, a cor vermelha, uma atividade irregular.

O estado de cada atividade representado pelo gerenciador depende do estado do documento que está associado a tal atividade (associação realizada pelo *link*). Um documento pode ter seu estado em um dos três tipos: "*frozen - congelado*", "*not frozen – não congelado*", "sem documento associado ou documento vazio" ou "*not frozen – não congelado* e atividade processada", os quais correspondem aos estados das atividades.

O documento recém criado que está sendo editado e ainda não está "*frozen*" indica que a atividade correspondente pode ser considerada "em andamento", ou seja, é um documento cuja edição e implementação ainda não terminou.

Uma atividade pode ser considerada irregular, quando possui um *link* apontando um documento e esse documento foi excluído do ambiente ou, ainda, no caso de existir um *link* na atividade considerada processada, e esse *link* foi excluído, e cria-se um novo, associado à essa atividade.

Quando ocorrer uma decisão entre quais atividades devem ser processadas e quais atividades devem ser rejeitadas, as não escolhidas pela decisão recebem o estado de descartada. No caso das atividades humanas e automatizadas estarem no estado "não processada", estas aparecem na página onde se encontra a opção do sistema gerenciador de *workflow*. As atividades humanas são consideradas processadas, apenas quando o ator responsável por tal atividade, as seleciona e clica no botão "*Performed*". Em relação às atividades automatizadas, elas apenas são representadas na página com seus atributos. A figura 3.11 representa a janela referente à opção das atividades humanas.



Fonte: (SCHEIDT, 2002)

Figura 3.11 - Janela das atividades humanas

A ferramenta de gerenciamento permite visualizar as atividades que podem ser processadas e as que não podem ser processadas naquele instante do projeto.

Assim como as atividades, as decisões também apresentam cores para a identificação de seus estados: verde representa uma decisão tomada, e azul representa decisão a ser tomada. O início e término de processo (*BeginEnd*) também utilizam cores para a representação de início de *workflow* (*begin* é representada pela cor verde) e término do projeto (*end* é representado pela cor azul).

Com as inclusões do sistema gerenciador de *Workflow* e do editor de *workflow*, algumas características foram observadas no ambiente SEA: organização dos processos, controle automático do estado dos processos, distribuição das atividades a serem realizadas, acompanhamento do trabalho, controle de consistência de informações e priorização de tarefas, solucionando, assim, a deficiência em relação ao gerenciamento das atividades constatadas no ambiente SEA.

3.3 Conclusão

A descrição do ambiente SEA foi necessária para entender o ambiente, o que ele disponibiliza e quais as funções oferecidas por ele e para entender onde foram acrescentadas funcionalidades para suprir algumas deficiências encontradas no ambiente que serão descritas no capítulo 6.

O entendimento do ambiente SEA ajuda a justificar o desenvolvimento deste trabalho, pois algumas funcionalidades não estão presentes nele, como por exemplo, a possibilidade de acompanhamento de atividades como inspeção, documento de especificação de requisitos, plano de testes e aplicação dos mesmos. A implementação dessas funções oferecerá ao ambiente maior organização, estruturação e registro das informações realizadas num projeto. Essa documentação foi semi-automatizada em forma de documentos textuais estruturados, suprindo, com isso, a carência de alguns documentos pertencentes ao processo. Com isso, pode-se fornecer subsídios para documentar as atividades gerenciais estabelecidas no Projeto de *Workflow*.

Apesar de o ambiente de desenvolvimento SEA fornecer suporte ao gerenciamento das atividades de um projeto, o ambiente não possui nenhum suporte para documentar tais atividades como inspeção, especificação de requisitos, planos de testes, formulário de aplicação de testes. O Projeto de *Workflow* tem, no entanto, necessidade de uma documentação semi-automatizada para facilitar o registro de suas atividades que demandam a "geração" de documentos textuais. Esse aspecto, motivou, portanto, a realização deste trabalho.

O próximo capítulo apresenta o modelo CMMI como contribuição parcial para construção de estrutura de documentos textuais.

4 CMMI – Modelo de Maturidade de Capacitação Integrado

O desenvolvimento de *software* de alta qualidade é uma preocupação freqüente das organizações que operam com engenharia de *software*. A produção de *software* envolve qualidade, segurança, manutenção e prazos por isso, empresas de desenvolvimento de *software* preocupam-se muito com a melhora dos seus processos (SANTANDER & VASCONCELOS, 2002). A partir de um processo de desenvolvimento adequado é provável que se obtenha um produto de boa qualidade.

Atualmente existem modelos que auxiliam na melhora do processo de desenvolvimento de *software.* Dentre os mais conhecidos, pode-se citar, o padrão ISO 9001 (*INTERNATIONAL ORGANIZATION FOR STANDARDIZATION*, 2004b), o padrão ISO/IEC 15504 (*INTERNATIONAL ORGANIZATION FOR STANDARDIZATION*, 2004a) e o *Capability Maturity Model Integration for Software* (CMMI) *Staged* (*SOFTWARE ENGINEERING INSTITUTE*, 2003a), que é um suporte para o objeto de estudo deste trabalho e trata do aspecto gerencial do processo. A escolha da proposta do trabalho baseado em CMMI justifica-se pelo fato de já terem sido adaptadas PAs de CMMI (SCHEIDT, 2002) no ambiente de desenvolvimento de *software* SEA com o objetivo de incluir ao ambiente qualidade no processo.

O CMMI é uma nova versão de CMM, que trata de Modelo de Maturidade de Capacitação Integrado, o CMMI. Esta nova versão do modelo promete corrigir e atualizar o modelo atual integrando os modelos existentes de CMM aplicados a diferentes disciplinas e, também compatibilizá-los com outros padrões como ISO 9001 e ISO/IEC 15504. O CMMI possui duas modalidades: a modalidade *Continuous* e a *Staged,* a qual será abordada neste trabalho.

Essa tecnologia era abordada em CMM, mas separadamente, pois, existem vários modelos CMM independentes. Para cada aplicação existia um modelo CMM referente à necessidade de seu uso. A nova terminologia estaria facilitando a redução da quantidade de treinamento necessário e reduzir o tempo gasto, trabalho, custos para uma organização.

A finalidade de um projeto CMMI é inserir no desenvolvimento de *software*, práticas aplicáveis a processos produtivos em geral, proporcionando, por meio de seus requisitos, gerenciamento de processos e melhoria de qualidade. A meta a ser alcançada utilizando esse modelo é a maturidade do processo de desenvolvimento (*SOFTWARE ENGINEERING INSTITUTE*, 2003a).

O CMMI permite melhorar a eficiência, retornar investimentos, aumentar a efetividade em processos utilizados por organizações que utilizam modelos e integram partes como engenharia de sistemas e engenharia de *software*, que são inseparáveis em desenvolvimento de *software* (HERNDON & et.al., 2003).

A representação *Continuous* pode ser aplicada a qualquer tipo de serviço dentro de uma organização, onde esta pretende melhorar seus processos. Já o projeto do modelo CMMI *Staged* surgiu da necessidade de melhorar a utilização do modelo CMM, pois este modelo ainda está sendo adotado por muitas organizações. Assim, foi proposto o modelo *Staged* para suprir algumas deficiências identificadas pelo *Software Engineering Institute* em relação ao modelo CMM (*SOFTWARE ENGINEERING INSTITUTE*, 2003a) podendo estas organizações adaptar o seu antigo modelo a nova versão para *software*, o CMMI *Staged*, pois é provável que até final de 2004 o modelo CMM seja extinto do mercado.

Este capítulo apresenta uma breve descrição do Modelo de Maturidade de Capacitação Integrado. A representação *Staged* será descrita mais especificamente, bem como suas características referentes a áreas de processo deste modelo que demandam documentação. A abordagem referente a representação *Staged,* é devido ao enfoque deste trabalho pois, além de estar inserindo qualidade ao ambiente SEA, a descrição de algumas PAs comprova a necessidade da documentação de atividades decorrentes do processo de desenvolvimento, justificando assim, a proposta do trabalho e será baseado no relatório do modelo fornecido pelo *Software Engineering Institute*.

4.1 Representações do Modelo CMMI

O modelo CMMI possui duas representações *Continuous* e *Staged*.

4.1.1 CMMI *Continuous*

No CMMI *Continuous*, o enfoque é semelhante à norma ISO/IEC 15504 devido à organização detalhada em categorias e processos. Objetiva a melhoria de cada área do processo direcionado à sua avaliação através dos 6 (0-5) níveis de maturidade (*SOFTWARE ENGINEERING INSTITUTE*, 2003b). Possui o mesmo conjunto de áreas de processo do modelo em estágios, sem seu agrupamento em níveis. Esta representação não será detalhada porque o objetivo do trabalho está associado à representação *Staged,* que é uma nova versão do CMM.

4.1.2 CMMI *Staged*

No CMMI *Staged* é estabelecida uma seqüência ordenada de melhoramentos com práticas de gerenciamento e processos. Nesta modalidade a qualidade do processo depende dos níveis de maturidade como CMM, porém, CMMI possui um conjunto de áreas de processo ligeiramente diferente do modelo CMM (em que sua denominação era Áreas Chave do Processo). A utilização do CMMI *Staged* também proporciona uma facilidade de migração dentro das organizações que adotam o CMM e querem passar adaptar suas práticas para o CMMI *Staged* (*SOFTWARE ENGINEERING INSTITUTE*, 2003a).

Esse modelo descreve os elementos-chave de um processo de desenvolvimento de *software* efetivo, onde organizações podem evoluir para um processo disciplinado e maduro, procurando orientar-se no gerenciamento do processo de desenvolvimento de *software* em busca da qualidade total.

O CMMI *Staged* possui práticas que auxiliam na implantação da melhoria contínua do processo de desenvolvimento do *software*. Esse modelo é baseado em pequenos passos que evoluem através dos cinco níveis crescentes de maturidade, e prioriza as ações a serem realizadas classificando-as em cada um destes cinco níveis.

O CMMI estabelece cinco (5) níveis crescentes de maturidade para controlar o processo de maturidade: nível 1 ou inicial, nível 2 ou Gerenciado (capacidade de gerenciar um projeto), nível 3 ou Definido (processo em desenvolvimento definido), nível 4 ou Gerenciado

Quantitativamente (histórico de desenvolvimento) e nível 5 ou Em Otimização (melhoria de processo - capacidade de interferir no processo) (*SOFTWARE ENGINEERING INSTITUTE*, 2003a).

Ressalta-se que cada um dos 4 níveis CMMI *Staged*, exceto o nível 1, consiste num grupo de atividades correlatas, chamadas Áreas de Processo. Os níveis de maturidade na categoria *Staged* prevêem uma ordem de agrupamento dos processos de melhorias em estágios.

4.2 Descrição do Modelo CMMI *Staged*

O CMMI *Staged* é composto de um conjunto de requisitos a serem atingidos na busca da qualidade. Esse conjunto de requisitos compreende um conjunto de objetivos a serem atendidos.

O modelo de melhoria no processo de *software* preocupa-se com a qualidade no gerenciamento do processo do desenvolvimento de *software* e não com a qualidade do produto final. O CMMI *Staged*, apesar de estar voltado para o processo, não trata o aspecto tecnológico do processo.

Este modelo auxilia as organizações identificarem possíveis deficiências que poderiam ser encontradas no produto final, podendo ser identificadas e corrigidas durante o seu processo de desenvolvimento, evitando possíveis erros durante o desenvolvimento do *software*.

O objetivo do CMMI *Staged* é ajudar as organizações a atingirem o maior grau de maturidade do processo melhorando seus produtos de trabalho. Esse grau de maturidade poderá auxiliar na qualidade do produto final, pois o CMMI *Staged* descreve os princípios e práticas de maturidade de processo de *software* auxiliando as empresas na melhoria da maturidade de seus processos, podendo passar de processos caóticos a processos maduros e disciplinados (HERNDON, 2003).

Cada um dos cinco níveis possui seus próprios requisitos, permitindo passar de um nível para outro quando todos os requisitos do nível anterior estiverem cumpridos. Cada nível de maturidade é, portanto, uma base para o nível seguinte, permitindo assim, uma implantação eficiente e eficaz em um programa de melhoria da qualidade no processo de desenvolvimento do *software*.

Em cada nível o ambiente precisa demonstrar que tem plena capacitação em determinadas áreas de processo, pois para uma organização obter certificação nível 2, o nível gerenciado, todas as áreas de processo relativas a funções gerenciais precisam estar implantadas. Para atingir o nível 3, o nível definido, as áreas de processo referentes a todos os procedimentos técnicos necessários para a execução de atividades técnicas, de treinamento e de revisão interna já devem estar plenamente capacitadas. Para passar ao nível gerenciado quantitativamente ou nível 4, são exigidas as áreas de processo correspondentes às formas de medir e avaliar cada procedimento técnico.

Uma organização só terá o nível 5 após todas as áreas de processo do nível 4 forem atingidas completamente. Finalmente, no nível mais elevado, áreas de processo específicas ancoram o processo de melhoria contínua com correção de problemas e evolução de processos - nível 5 ou em Otimização (*SOFTWARE ENGINEERING INSTITUTE*, 2003a).

4.2.1 Estrutura da Representação CMMI *Staged*

Na representação *Staged*, cada nível de maturidade, é organizado por um conjunto de áreas de processo, as quais são organizadas por objetivos específicos e objetivos genéricos. Os objetivos específicos são organizados por práticas específicas e os objetivos gerais são organizados por práticas gerais que por sua vez são organizadas por características comuns. Cada PA possui seus requisitos a serem cumpridos para estar em conformidade com o CMMI.

A estrutura externa modelo CMMI categoria *Staged* é demonstrada na figura 4.1.

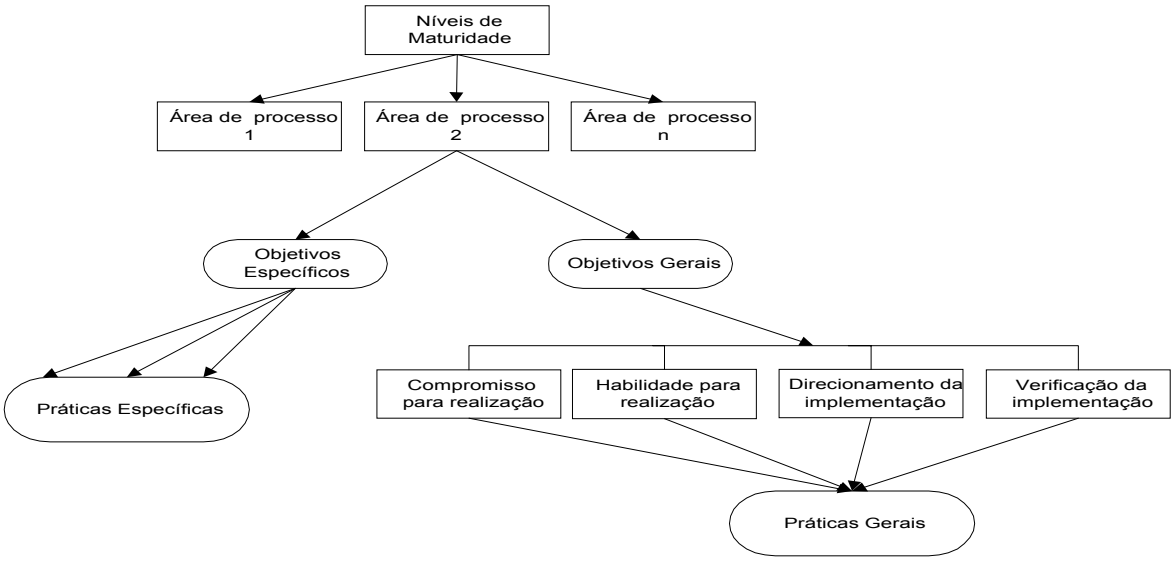


Figura 4.1 - Estrutura do modelo CMMI categoria *Staged*

A figura 4.1 demonstra como está organizada a estrutura do Modelo de Maturidade de Capacitação Integrado, representação *Staged*, internamente.

O próximo item apresenta algumas PAs do Nível 2 de CMMI *Staged* em que seus requisitos demandam documentação.

4.3 Áreas de Processo do Modelo CMMI *Staged*

Cada nível do CMMI é caracterizado por um conjunto de áreas de processo (PAs), que por sua vez, possuem requisitos a serem cumpridos para ser compatível ao CMMI. Sendo que, as áreas de processos são definidas em termos de práticas genéricas e específicas para atingir objetivos genéricos e específicos. A seguir são listadas as áreas de processo de cada nível de CMMI.

O Nível 1 não possui áreas de processo. As organizações neste nível, são caracterizadas pela sobrecarga de atividades dos membros da organização, pelo abandono de projetos cada vez que se encontram numa crise e por não conseguir repetir suas experiências em projetos futuros devido ao insucesso dos mesmos.

4.3.1 Áreas de Processo no Nível 2 (Gerenciado)

Este nível possui a capacidade de gerenciar um projeto, seguindo as estimativas, planejamento e cronograma estabelecidos. As informações são todas documentadas em nível de projeto. A capacidade de reconhecer problemas e corrigi-los na medida em que estes ocorrem é identificada nesse nível.

As áreas de processo presentes no nível dois são:

- Gerenciamento de Requisitos;
- Planejamento de Projeto;
- Monitorização e Controle de Projeto;
- Gerenciamento de Contrato com Fornecedor;
- Medição e Análise;
- Garantia de Qualidade de Processo e Produto;
- Gerenciamento de Configuração.

4.3.2 Áreas de Processo do Nível 3 (Definido)

Neste nível inclui-se a preocupação com o aspecto tecnológico integrado ao aspecto gerencial definido no nível 2. O nível 3 fornece subsídios para que cada organização adapte o processo global de acordo com suas características e necessidades, seguindo o foco do processo padrão definido na organização.

Partindo disso, o nível 3 é caracterizado pelos processos bem compreendidos e definidos de acordo com padrões, procedimentos, ferramentas e métodos. As áreas de processo do nível três são listadas a seguir.

- Desenvolvimento de Requisitos;
- Solução Técnica;
- Integração de Produto;
- Verificação;
- Validação;

- Foco no Processo da Organização;
- Definição do Processo da Organização.
- Treinamento Organizacional;
- Gerenciamento de Projeto Integrado;
- Gerenciamento de Risco;
- Análise de Decisão e Solução.

4.3.3 Áreas de Processo do Nível 4 (Quantitativamente Gerenciado)

Neste nível as organizações possuem a capacidade de interpretar os resultados pelos números e estabelecer decisões (planejamento, gerência) baseadas nesses resultados. Essas decisões são tomadas através de estimativas mais precisas. As estimativas podem ser obtidas com aplicação de métricas.

Caso os objetivos estabelecidos estejam sendo desviados ao longo do processo, isso é imediatamente identificado e, quando isso ocorre, são aplicadas ações corretivas. As áreas de processo do nível 4 são expostas seguir.

- Desempenho do Processo Organizacional;
- Gerenciamento Quantitativo de Projeto.

4.3.4 Áreas de Processo do Nível 5 (Em Otimização)

Neste nível é possível as organizações avaliarem seus processos identificando seus pontos fracos e pontos fortes. Sendo que pode-se eliminar os problemas identificados nos pontos fracos. No nível 5, as organizações possuem a responsabilidade de aumentar a qualidade nos seus processos através de ações pró-ativas. Deve-se manter continuamente a melhora dos seus processos.

A seguir são expostas as áreas de processo do nível 5.

- Inovação Organizacional e Difusão;
- Ação Causal e Solução.

4.4 DOCUMENTAÇÃO TEXTUAL ASSOCIADA A ÁREAS DE PROCESSO DE CMMI

A documentação das atividades que são realizadas durante o processo de desenvolvimento é uma tarefa essencial dentro de uma organização, mas nem todas as organizações desenvolvedoras de *software* fazem uso desse procedimento. Isso acontece normalmente, devido ao fato de as empresas não terem muito conhecimento de como fazer para melhorar seus processos, ou também pelo fato do curto período de tempo em que as empresas definem seus cronogramas.

A importância da documentação é justificada, pois todas as áreas de processo, sem exceção, demandam documentação textual. Embora o modelo não especifique como tais atividades devem ser documentadas, este fornece subsídios para que isso seja cumprido. Assim, a proposta do trabalho foi direcionada para fornecer subsídios com a finalidade de apoiar requisitos de CMMI em uma ferramenta CASE (*Computer Aided Software Engineering*) por meio de uma estrutura de documentos textuais.

A seguir são descritas algumas PAs do nível dois e três que necessitam documentar suas atividades implantadas de acordo com os requisitos definidos. Apesar de todas as atividades necessitarem do registro de suas informações, apenas algumas PAs serão descritas, justificando o processo de documentação em seus produtos de trabalho, pois são tratadas apenas algumas em um estudo de caso.

4.4.1 Áreas de Processo do Nível 2

O nível 2 de CMMI é ressaltado pelo seu desenvolvimento "gerenciado" e é caraterizado pelos seguintes aspectos:

- As organizações devem possuir capacidade e subsídios para gerenciar um projeto;
- As organizações devem possuir no desenvolvimento: estimativas, planejamento, compromissos definidos (e documentados no nível de projeto);
- Nesse nível os problemas encontrados são reconhecidos e devem ser corrigidos na medida em que estes ocorrem.

A partir dessa abordagem, as organizações têm plena certeza que os projetos são gerenciados, que os processos são planejados, executados, medidos e controlados, pois quando as práticas do nível dois são aplicadas, projetos são gerenciados e executados de acordo com seus planos documentados.

A seguir são apresentadas PAs que necessitam de documentação em relação ao seu planejamento e execução. Apesar de expor apenas duas PAs do nível dois, há outras, senão todas, que de alguma forma necessite documentar suas informações.

4.4.1.1 Monitorização e Controle de Projeto

O objetivo dessa PA é fornecer um entendimento do progresso do projeto além de aplicar ações corretivas quando o projeto desvia do seu planejamento.

Nessa área de processo, a base é monitorar as atividades e aplicar ações corretivas caso o projeto desvie do que foi definido. Para que estas atividades sejam possíveis, deve haver uma documentação a ser comparada mais adiante em relação ao que foi realizado e o que foi proposto inicialmente. E, caso necessite de ações que modifiquem possíveis problemas, estes devem ser analisados e corrigidos ou replanejados, sempre baseado na proposta inicial documentada.

4.4.1.2 Garantia de Qualidade do Processo e Produto

O propósito desta PA é fornecer subsídios para o melhoramento na parte administrativa e gerencial de processos e produtos de trabalho associados.

Nesta PA, utiliza-se documentação para identificar e registrar as não conformidades dos processos, para que ações corretivas possam ser aplicadas, baseadas numa informação coerente.

4.4.2 Áreas de Processo do Nível 3

O Nível 3 do CMMI é caracterizado pela integração entre o aspecto tecnológico e o aspecto gerencial implantado no nível 2. Os projetos neste nível, estabelecem que seus processos sejam bem definidos por meio da adaptação do conjunto de processos padrão da organização, de acordo com um caminho para implantação.

A seguir são apresentadas duas PAs do nível 3, que estão contempladas no estudo de caso realizado neste trabalho em relação às atividades relacionadas aos requisitos estabelecidos.

4.4.2.1 Verificação

A finalidade desta PA é assegurar que os produtos de trabalho analisados estão de acordo com o que foi especificado inicialmente, envolvendo preparação para verificação, performance da verificação e identificação de ações corretivas.

Aplicar um tipo de verificação é inerentemente um processo incremental porque ocorre através do desenvolvimento dos produtos de trabalho, iniciando com uma verificação dos requisitos e o progresso através da verificação dos produtos de trabalho envolvidos ao longo do processo.

Uma forma de implantar esta PA é aplicando a atividade de inspeção aos produtos de trabalho. Essa atividade é utilizada em vários produtos de trabalho e sempre deve ser documentada.

4.4.2.2 Validação

O objetivo desta PA é demonstrar que um produto de trabalho está em conformidade com o que foi especificado. Os produtos de trabalho podem ser requisitos de projeto, protótipo e quando forem definidos, devem ser baseados no melhor caminho de como esses produtos de trabalho poderão satisfazer as necessidades dos usuários.

A aplicação da PA Validação deve apresentar atividades convenientes de acordo com os critérios dos produtos de trabalho analisados. As questões referentes à validação podem ser descobertas ao longo do projeto usando seus produtos de trabalho. Mas, para isso, atividades de validação devem ser documentadas, como por exemplo, a aplicação de testes em relação aos produtos de trabalhos definidos.

Mas, para que testes sejam aplicados, primeiramente deve ser elaborado um documento de plano de teste e, posteriormente, um documento registrando a aplicação dos casos de teste definidos no plano, ou seja, um outro documento de aplicação dos casos de testes. Essa PA, mais uma vez, comprova a necessidade de registrar informações associadas a produtos de trabalho.

À medida que as questões são identificadas, são encaminhadas aos processos associados, como desenvolvimento de requisitos, solução técnica ou monitoração de projeto e controle para uma resolução.

4.5 CMMI versus *Rational Unified Process*

O modelo CMMI sugere práticas para melhoria do processo de seus produtos de trabalho, ou seja, apenas define o que deve ser feito para melhoria do processo, mas não estabelece como as práticas devem ser implantadas (*SOFTWARE ENGINEERING INSTITUTE*, 2003a).

Nesse contexto, tentou-se "casar" CMMI com *Rational Unified Process* (RUP), devido ao RUP ser um processo unificado e nele estar disponível o "como" tais informações podem ser concretizadas. Todavia, modelos de documentos são definidos no RUP.

Neste trabalho a adoção do CMMI justifica-se pela aplicação de requisitos que podem melhorar a qualidade dos recursos disponíveis no ambiente de desenvolvimento de *software* SEA e para reforçar a importância do registro das informações de atividades realizadas durante o processo.

Assim, os modelos de documentos adotados no trabalho para registrar certas informações que são descritas no capítulo 6, são baseados no RUP (KRUCHTEN, 2001). Optou-se por seguir modelos de documentos do RUP por ser um processo bem definido e, também, segundo autores, é dito que: "se for aplicado todo o RUP dentro de uma empresa, esta terá certificado nível 5 de CMMI" (KRUCHTEN, 2001).

4.6 Conclusão

Este capítulo apresentou brevemente o modelo de maturidade de capacitação integrado, descrevendo sucintamente os níveis de maturidade e suas áreas de processo.

A descrição deste capítulo justifica-se pela comprovação da importância de registrar informações, relacionado à documentação textual. Optou-se por descrever algumas áreas de processo, apenas as contempladas nos *templates*, pois o objetivo deste capítulo é de justificar a importância de uma estrutura de documentação textual dentro de uma ferramenta CASE com a finalidade de melhorar seus produtos de trabalho de acordo com um processo compatível ao CMMI *Staged*, melhorando a qualidade dos mesmos. E, também, justificar a utilização de modelos de documentos do RUP.

Todas as PAs utilizam meios para registrar as suas informações, pois, caso contrário, como seria possível comprovar certas informações e sugestões de modificação quando algo não está em conformidade com o que foi estabelecido?

O próximo capítulo apresenta alguns trabalhos correlatos encontrados na linha de pesquisa deste trabalho.

5 Trabalhos Correlatos

O presente trabalho possui algumas semelhanças com outros modelos desenvolvidos seguindo a mesma linha de pesquisa. Assim, estudos foram realizados sugerindo algum método para tratar o registro e a organização das informações textuais por meio de documentos durante o processo de desenvolvimento de *software.* Técnicas no gerenciamento no processo de desenvolvimento de *software* como aspecto importante para auxiliar na melhora do processo de desenvolvimento são apresentados neste capítulo.

5.1 Organização de Informações Textuais

Estudo realizado por (FIORINI et. al., 1998) com o título "Organizando Processos de Requisitos" propõe um esquema básico para organização de processo de requisitos com ênfase na descrição de padrões de processos. Os requisitos formam a base para o planejamento, o acompanhamento do desenvolvimento e a aceitação dos resultados dos projetos de *software*, atividades presentes no processo de desenvolvimento de *software*. O principal objetivo do processo de definição de requisitos é finalizar com sucesso um acordo entre quem solicita e quem desenvolve o *software*. Devido ao fato de a organização e definição de requisitos estarem diretamente relacionados com a Gerência de Requisitos, utilizou-se como base o modelo CMM. Nele está definida, a necessidade da Gerência de Requisitos (apresentado no capítulo 4). Os processos de requisitos necessários para que uma organização possa defini-los e gerenciá-los precisam estar devidamente organizados. Para isso, foi utilizada uma forma de documentar os processos segundo três pontos de vista (processo padrão, padrão de processo e processo solução) e seguindo três níveis de detalhamento (processo indivíduo, processo em família e processo em comunidade), possibilitando, assim, maior organização e entendimento dos processos, permitindo a adaptação dos mesmos às características de cada projeto.

O Apoio à documentação de engenharia reversa de *software* por meio de hipertexto é um trabalho de pesquisa desenvolvido por (FELTRIN, 1999) na Universidade de São Paulo - USP, capaz de recuperar informações perdidas durante a fase de desenvolvimento do *software* e de documentar seu estado atual inserindo, assim, o conceito de Engenharia Reversa, devido

à documentação incompleta e informal encontrada às vezes pelo engenheiro de *software*. O objetivo principal desse trabalho foi a identificação de uma estrutura de hiperdocumentos para dar apoio à documentação solicitada durante o processo de engenharia reversa de *software*. Realizado por um levantamento adequado dos requisitos necessários num hiperdocumento, para que suporte à documentação de engenharia reversa, foi definido um conjunto de *links* e estrutura de nós. Os requisitos para desenvolver o hiperdocumento foram extraídos por meio de uma autodocumentação do sistema hipermídia, o SASHE (Sistema Hipermídia para Ensino), e a Engenharia Reversa foi baseada no método de Engenharia Reversa *Fusion*-RE/I.

5.2 Gerência do Processo no Desenvolvimento de *Software*

A dissertação realizada na Universidade Federal do Rio Grande do Sul - UFRGS, "Uso de sistema de gerência de *workflow* para apoiar o desenvolvimento de *software* baseado no processo unificado da *rational* estendido para alcançar níveis 2 e 3 do modelo de CMM" (MANZONI, 2001) foi desenvolvido com a finalidade de apoiar o processo de desenvolvimento de sistemas, identificando os problemas e as vantagens do uso destes sistemas para modelar processos de *software*. A modelagem do processo de *software* é baseada no *Rational Unified Process* (RUP) com adaptações para atingir o modelo de avaliação de Processo *Capability Maturity Model* (CMM). O modelo de processo desenvolvido poderá auxiliar as organizações na formalização e melhoria de seus processos de desenvolvimento de *software*.

A dissertação realizada na Pontifícia Universidade Católica do Paraná - PUCPR, "Uma ferramenta automatizada de suporte ao processo de gerenciamento de requisitos" desenvolvido por (CORDEIRO, 2002) enfatiza o gerenciamento de requisitos de *software*, como um dos elementos principais de um processo de desenvolvimento de *software* efetivo. O objetivo principal desse estudo é mostrar a potencialidade de um processo efetivo de gerenciamento de requisitos como forma de melhorar a especificação solicitada pelo cliente e o que lhe é entregue como produto final. O gerenciamento de requisitos nesse trabalho é baseado no modelo CMMI (*Capability Maturity Model Integration*) e pelas normas ISO/IEC 12207 e 15504. Para a modelagem do gerenciamento de requisitos, utiliza-se uma metodologia orientada a processo a qual é proposta pelo Ambiente da Inteligência da Realidade (AGIR). A partir dessa modelagem, foi desenvolvida uma ferramenta para

automação do processo de gerenciamento de requisitos de uma organização. Os resultados decorrentes desse gerenciamento são analisados, assegurando que um processo cuidadoso de especificação e gerenciamento de requisitos contribui para um produto de *software* de qualidade.

O Ambiente de Desenvolvimento de *Software*, TransCoop (CORDENONZI, 2000) foi desenvolvido na Universidade Federal do Rio Grande do Sul direcionando seu estudo na área de suporte ao desenvolvimento de *software* e utilizando técnicas de gerência de projeto (baseados em *workflow*), trabalho cooperativo e banco de dados. Esse projeto propõe um modelo conceitual para definição e acompanhamento do processo de desenvolvimento. Inclui características referentes ao padrão ISO/IEC 9126, para qualidade do produto de *software,* e de práticas-chave do CMM, para qualidade do processo de *software*. Essas práticas são mapeadas por diagramas de classes e diagrama de transição de estados de acordo com restrições de integridade. É necessário salientar que apenas o nível dois de CMM foi mapeado nesta dissertação.

O Projeto ADS (MIAN et. al., 1999) desenvolvido na Universidade Federal do Espírito Santo, pretende tratar uma deficiência encontrada em ferramentas e ambientes de desenvolvimento de *software*. O objetivo principal desse projeto foi a construção de um ambiente integrado destinado à experimentação da Engenharia de *Software*. O ADS foi distribuído por vários trabalhos, cada um tratando do desenvolvimento de um módulo ou ferramenta específica de cada ambiente. Utiliza a plataforma Orientada a Objetos através da Linguagem de programação Java e Banco de Dados Relacionais. O propósito desse projeto ADS era obter um ambiente integrado que fornecesse serviços de infra-estrutura, proporcionando a integração de ferramentas individuais ao longo das seguintes dimensões: controle, dados, interface com o usuário e conhecimento. É baseado no trabalho desenvolvido em (MIAN, 2001), o qual enfoca o acompanhamento e execução de projetos, de maneira a automatizar o processo de desenvolvimento e fornece apoio ao gerente de projetos no monitoramento do desenvolvimento de *software*. Assim, a meta do projeto ADS é apoiar todo o ciclo de vida e oferecer suporte para o gerenciamento de projetos, estabelecendo uma estrutura técnica para integração e controle. Com essa integração, ferramentas CASE de diferentes paradigmas, que apóiam várias fases do ciclo de vida do processo, podem ser incorporadas ao ADS, mantendo uma integridade entre os produtos gerados por elas individualmente.

5.3 Conclusão

Relacionando o estudo de (FIORINI, et. al., 1998), "Organizando processos de Requisitos" ao novo ambiente SEA, no qual é proposta a Estrutura de Documentos Textuais, pode-se observar que os dois estudos possuem estrutura para registro de informações e formam base para o planejamento e acompanhamento do desenvolvimento de *software*. Enquanto, neste trabalho, tenta-se adaptar práticas do Modelo de Maturidade de Capacitação Integrado (CMMI) ao ambiente SEA, o estudo realizado por (FIORINI, et. al., 1998) é baseado no Modelo de Maturidade de Capacitação (CMM), pois a definição de requisitos está diretamente relacionada com a Gerência de Requisitos, uma das KPAs do nível dois de CMM (estudo realizado antes de ter sido publicado o CMMI). Apesar de ser proposta no estudo citado, uma organização no processo de requisitos, o SEA é mais completo que apenas uma organização nos processos. Este ambiente, por ser de desenvolvimento de *software*, também possui gerenciamento nas atividades desenvolvidas a partir dele e, agora, possui uma estrutura de documentos textuais para registro das informações obtidas pelas atividades realizadas num projeto. A estrutura de documentos textuais disponível no SEA justifica-se pelos PAs de CMMI que demandam documentação. Essa estrutura de documentos textuais trata o problema de requisitos e muito mais, como, por exemplo, a inspeção de um documento de especificação de requisitos, especificação OO, ou seja, apenas a contribuição deste trabalho ao ambiente SEA já é mais ampla do que o estudo realizado por (FIORINI, et. al., 1998).

Já o "Apoio à documentação de engenharia reversa de *software* por meio de hipertexto", desenvolvido por (FELTRIN, 1998) possui apenas suporte para recuperar informações que normalmente são perdidas durante a fase de desenvolvimento do *software* para poder documentar o seu estado atual. Não foram encontrados subsídios em relação a algum modelo de processo e nem subsídios para prover registro das atividades em forma de documentos como no ambiente SEA. O levantamento dos dados é realizado pelos requisitos necessários num hiperdocumento definidos por um conjunto de *links* e uma estrutura de nós. O Hiperdocumento foi desenvolvido a partir de uma autodocumentação do SASHE e baseada no método de Engenharia Reversa *Fusion*-RE/I. Em relação à proposta de Estrutura de Documentos Textuais no ambiente SEA, esses dois estudos possuem uma única similaridade: a inclusão de documentos textuais para registro das informações, como objetivo principal dos dois trabalhos.

O estudo realizado por (MANZONI, 2001), propõe a utilização de sistema de gerência de *workflow* para apoiar o desenvolvimento de *software* baseado no processo unificado da *rational*. Esse estudo tem a finalidade de apoiar o processo de desenvolvimento de sistemas, identificando os problemas e as vantagens da utilização deles para modelar os processos de *software*. Em seu gerenciamento, utiliza sistemas *workflow*, como o ambiente SEA utiliza-o para gerenciar suas atividades. Como modelo de processo, utiliza CMM, semelhante em relação ao ambiente SEA que se baseia no CMMI *Staged*. A modelagem no processo é baseada no *Rational Unified Process* (RUP) assim como os *templates* de documentos implantados no SEA. Além disso, o SEA dispõe de funcionalidades que auxiliam no desenvolvimento de *software*, como especificação UML, e práticas inseridas ao ambiente que melhoram a qualidade dos seus processos. Inclui, ainda, ferramentas de verificação de consistência que analisam as informações geradas em cada especificação criada no SEA.

A ferramenta desenvolvida por (CORDEIRO, 2002) é referente ao suporte no processo de gerenciamento de requisitos. Na ferramenta de (CORDEIRO, 2002), o gerenciamento de requisitos é baseado no modelo CMMI (*Capability Maturity Model Integration*) e pelas normas ISO/IEC 12207 e 15504, sendo que a modelagem do gerenciamento de requisitos é construída por meio de uma metodologia orientada a processos. As novas funcionalidades do ambiente SEA são baseadas em práticas necessárias do CMMI implantadas por meio de uma estrutura de documentos textuais. Essa estrutura de documentos como já comentado anteriormente trata o problema de requisitos, fornece suporte para inspecionar documentos, organizar os planos de teste, registrar aplicação dos casos de teste definidos no plano de teste. Possui, também, uma verificação da coerência das informações que estão sendo estabelecidas nessa estrutura de documentos textuais.

O Ambiente de Desenvolvimento de *Software*, o TransCoop (CORDENONZI, 2000) é direcionado ao suporte de desenvolvimento de *software*. Como o ambiente SEA, o TransCoop também utiliza técnicas de projeto baseadas em sistemas de *workflow*. Outra característica do TransCoop é a utilização para o acompanhamento do processo de desenvolvimento, de características do padrão ISO/IEC 9126 e o modelo CMM. Já as novas funcionalidades introduzidas ao ambiente se baseiam em práticas do CMMI.

O projeto ADS desenvolvido por (MIAN et. al., 1999) propõe a construção de um sistema integrado referente à experimentação da Engenharia de *Software*. Esse projeto foi desenvolvido por módulos independentes que foram, mais tarde, integrados. Como o ambiente SEA, o Projeto ADS utiliza plataforma orientada a objetos, desenvolvido, porém, na linguagem Java, e utiliza Banco de Dados Relacionais. O SEA foi desenvolvido na linguagem de programação *Smalltalk,* totalmente orientada a objetos. O Projeto ADS permite o acompanhamento e execução dos projetos, suportando automatizar o processo de desenvolvimento e monitoramento do desenvolvimento de *software*, assim como o ambiente SEA.

Várias particularidades nos trabalhos descritas são semelhantes ao ambiente SEA. Como se pôde observar no capítulo anterior, esse ambiente suporta criação de especificação OO, Interface de Componentes, *CookBook* Ativo, Projeto de *Workflow* e uma Estrutura de Documentos Textuais adicionada ao ambiente, que é proposta deste trabalho.

O ambiente SEA também possui gerenciamento de projeto, assim como alguns trabalhos citados. Para tratar esse gerenciamento, utilizam-se técnicas de *workflow*. No entanto, o modelo no processo utilizado para acompanhamento e monitoramento das atividades, é baseado no modelo CMMI (SCHEIDT, 2002), também identificado em alguns estudos apresentados neste capítulo. Desenvolvido na Linguagem *Smalltalk*, utiliza conceitos de orientação a objetos herdados da própria linguagem de programação.

Os *templates* dos documentos do estudo de caso pertencentes à Estrutura de Documentos Textuais são baseados no RUP, com práticas de CMMI inseridas a essa nova especificação.

O ambiente SEA também dispõe de ferramentas que verificam a consistência das informações definidas nas especificações do ambiente.

Do capítulo 2 ao 5, foi apresentado o embasamento teórico como alicerce para o desenvolvimento deste trabalho. Os capítulos 6 e 7 apresentam a contribuição referente às funcionalidades adicionadas ao ambiente de desenvolvimento de *software* SEA.

6 Documentos Textuais Estruturados no Ambiente SEA e Regras de Consistência

Este capítulo apresenta os documentos textuais estruturados implantados no ambiente SEA e as regras de consistência adotadas nesses documentos.

Os documentos textuais tratados no ambiente SEA são apenas *templates* (modelos estruturados), ou seja, modelos com informações necessárias em processo de desenvolvimento organizado e efetivo. Esses *templates* são baseados em *templates* adotados no *Rational Unified Process* (KRUCHTEN, 2001).

A idéia de inserir documentos textuais estruturados no ambiente SEA deve-se ao fato de que o ambiente não oferecia registro de informações como, por exemplo, inspeção, documento de especificação de requisitos e caso uma dessas atividades fosse exigida e estabelecida no projeto de *workflow*, quando cumpridas tais atividades, apenas era informado que tal atividade havia sido realizada, mas não havia o registro completo de tal. Com essa estrutura, foi introduzida melhoria na qualidade do processo de desenvolvimento de *software* dentro de uma ferramenta CASE.

Com isso, identificou-se PAs de CMMI que exigissem subsídios para melhorar o processo em relação à documentação das informações realizadas em um processo de desenvolvimento com a finalidade de melhorar a sua qualidade.

Esses documentos poderiam ter sido desenvolvidos em formato HTML (*HyperText Markup Language*), XML (*Extensible Markup Language*), ou arquivos de dados de programas como *Word*. No entanto, no trabalho aqui descrito, documentos textuais estruturados foram desenvolvidos estruturalmente, ou seja, possuem uma estrutura de documentos definida, onde devem ser estabelecidas as informações de cada documento, sendo que a preocupação em organizar tais documentos é desnecessária.

Um outro diferencial da estrutura de documentos textuais criada ao ambiente SEA é a avaliação de consistência através de ferramentas, fornecidas pelo OCEAN, também inseridas

ao ambiente SEA que tem a finalidade de conferir a consistência dos documentos gerados, informando se o documento atual está inconsistente ou não.

A seguir são descritos os documentos textuais implantados no ambiente SEA e as respectivas regras de consistência.

6.1 Modelos de Documentos Textuais Estruturados

Os documentos tratados no ambiente SEA são: documento de especificação de requisitos, documento de plano de testes, formulário de aplicação de testes e formulário de inspeção. Existem documentos como os que estão sendo implantados ao SEA que possuem outras informações, mas os documentos estabelecidos são baseados em documentos definidos no RUP com suas informações, sendo que as informações desses documentos foram tratadas (como pr exemplo, conceitos, atributos) seguindo a estrutura do SEA.

A relação entre CMMI e os documentos adicionados ao SEA serve para justificar a importância de documentar informações realizadas durante o processo.

É importante ressaltar que os documentos textuais estruturados inseridos ao ambiente SEA são *templates*, e que as informações a serem definidas estão estruturadas, e os dados em si precisam ser informados pelo usuário do ambiente. Os *templates* definidos são apenas um ensaio para inclusão de documentos estruturados no SEA, com a possibilidade de adicionar novos documentos (trabalhos futuros).

Segundo a terminologia utilizada no *framework* OCEAN, adotou-se na estrutura de documentos textuais que os documentos são tratados como modelos que referenciam conceitos, ou seja, as “seções” dos documentos foram definidas como conceitos (definição adotada pelo ambiente SEA) e as informações pertencentes a cada “seção” do documento, tratadas como atributos.

6.1.1 Documento de Especificação de Requisitos

Como mencionado, cada documento é um *template* com informações definidas como necessárias para registrar informações em atividades realizadas relacionadas a cada documento.

O documento de especificação de requisitos é um documento em que são agrupadas informações necessárias dos requisitos que serão tratados num projeto de desenvolvimento, tentando suprir as necessidades de um usuário (KRUCHTEN, 2004).

A seguir, são apresentados os conceitos e os respectivos atributos de um documento de especificação de requisitos.

O documento de especificação de requisitos definido no estudo de caso foi organizado nas seguintes seções, cada uma definida como um conceito de OCEAN:

- **Identificador do documento**;
- **Histórico do documento**;
- **Introdução**;
- **Visão geral do sistema**;
- **Conjunto de requisitos**.

O diagrama da figura 6.1 representa a organização do documento de especificação de requisitos e as seções que fazem parte do mesmo.

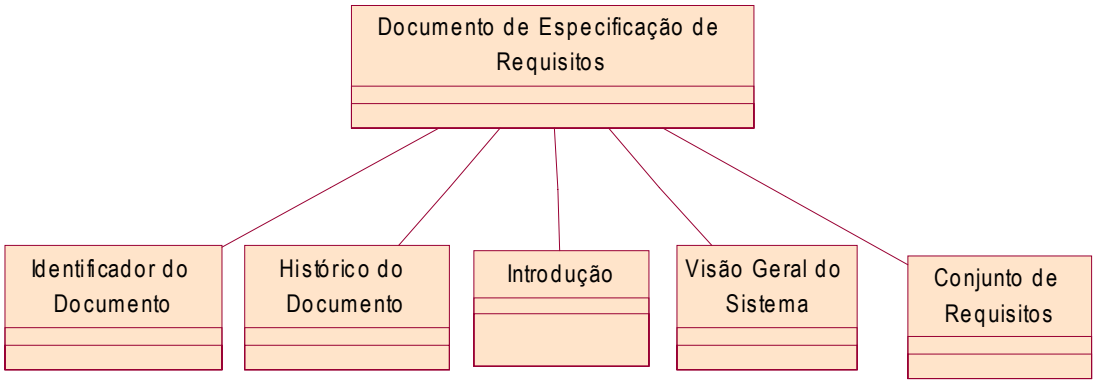


Figura 6.1 - Organização do Documento de Especificação de Requisitos

6.1.1.1 Conceito IdentificadorDoc

Este conceito é representado por um identificador do documento, ou seja, informações contidas na capa de um documento. Este é responsável por fornecer informações a serem definidas referentes à primeira seção de um documento, ou seja, informações que identifique, empresa, autor, projeto, por exemplo.

O conceito IdentificadorDoc possui os seguintes atributos:

- **Identificação da Empresa**;
- **Projeto:** identificação do projeto;
- **Aplicação - Parte do Projeto:** Identificação do *software* especificado;
- **Tipo de Documento (padrão da empresa)**;
- **Identificador do Documento (padrão da empresa)**;
- **Versão do Documento**;
- **Data da Conclusão do Documento**.

6.1.1.2 Conceito HistoricoDocumento

Este conceito é um histórico de um documento e agrega uma coleção de itens de históricos, outro conceito que possui seus atributos específicos. Um histórico do documento é uma seção de um documento de especificação que permite verificar o histórico do documento corrente através de algumas informações a serem definidas.

A coleção do HistoricoDocumento é apresentada automaticamente, assim que o conceito ItemHistorico for referenciado para preencher novo item.

6.1.1.2.1 Conceito ItemHistorico

Este conceito agrega atributos que formam um item de histórico do documento, ou seja, as informações que fazem parte de um histórico do documento. O conceito ItemHistorico possui os seguintes atributos:

- **Versão do Documento**;
- **Autor (es) do Documento**;
- **Data do Documento**;
- **Ação praticada**;
- **Esforço (em horas, especificado pelo autor)**.

6.1.1.3 Conceito Introducao

Este conceito contém informações necessárias de uma introdução de um documento e possui a finalidade de relatar informações sobre o que está sendo tratado em relação ao documento ao qual pertence à introdução. Esse conceito agrega os seguintes atributos:

- **Objetivo:** definição dos objetivos da especificação de requisitos, em linhas gerais, da aplicação a ser desenvolvida;
- **Descrição:** descrição do que será desenvolvido, siglas, abreviações e conceitos do problema citado ao longo da especificação;
- **Informações Gerais:** como referências bibliográficas, livros, normas...;
- **Localização:** referência do presente documento de requisitos no sistema de armazenamento.

6.1.1.4 Conceito Visão Geral do Sistema

Este conceito tenta fornecer informações que possam relatar estas em relação à arquitetura da aplicação a ser desenvolvida. O conceito Visão Geral do Sistema agrega os seguintes atributos:

- **Arquitetura do Sistema:** descrição do sistema em que a aplicação em desenvolvimento será inserida. Pode-se explicar todos os elementos externos que interagem com a aplicação;

- **Arquitetura da Aplicação:** estrutura modular da aplicação em desenvolvimento, caso essa informação esteja disponível no momento da especificação de requisitos;
- **Premissas de Desenvolvimento:** deve definir o que é assumido para iniciar o desenvolvimento - linguagem alvo, banco de dados, abordagem de desenvolvimento.

6.1.1.5 Conceito ConjuntoRequisitos

Este conceito agrega uma coleção de requisitos pertencentes a um documento de especificação de requisitos. O conceito Requisitos, é outro conceito referenciado pelo conceito ConjuntoRequisitos, que agrega as informações que são identificados por um atributo que representa a lista dos requisitos no conceito ConjuntoRequisitos.

Portanto, o conceito ConjuntoRequisitos possui um único atributo, a relação de requisitos gerada pelo conceito Requisito que é responsável por fornecer as informações a serem providas.

Este conceito possui três instâncias: conjunto de requisitos funcionais, conjunto de requisitos de interface e conjunto de restrições de projeto (requisitos não-funcionais). Cada instância agrega tantas instâncias de requisito quanto necessário.

6.1.1.5.1 Conceito Requisito

O Conceito Requisito é referenciado pelo conceito ConjuntoRequisitos e permite com que seja definido o requisito e sua descrição, o qual faz parte do projeto a ser desenvolvido. Este conceito agrega os seguintes atributos que representam as informações presentes no mesmo:

- **Identificador do Requisito:** identificação do requisito, por exemplo, requisito funcional 01;

- **Descrição do Requisito:** descrição da funcionalidade da aplicação. Pode ser representado por texto estruturado, por exemplo.

A seguir são descritas as instâncias do conceito ConjuntoRequisitos, isto é, cada vez que uma instância desse conceito agregar uma instância do conceito Requisito, informações dessas instâncias são definidas.

Instância Conjunto de Requisitos Funcionais:

- **Identificador do Requisito Funcional:** identificação do requisito funcional, por exemplo, requisito 01;
- **Descrição do Requisito Funcional:** descrição da funcionalidade da aplicação. Pode ser representado por texto estruturado, por exemplo.

Instância Conjunto de Requisitos de Interface:

- **Identificador do Requisito de Interface:** identificação do requisito de interface;
- **Descrição do Requisito de Interface:** descrição dos aspectos de interface relacionados ao desenvolvimento da aplicação, como, por exemplo, interface com usuários, relação de *APIs* (*Application Program Intreface*), interface com outros sistemas.

Instância Conjunto de Restrições de Projeto:

- **Identificação da Restrição:** identificação do requisito não-funcional;
- **Informações de Restrição de Projeto:** informações de requisitos não-funcionais como, por exemplo, suporte ao desenvolvimento (*UML*, *Delphi*, *Oracle*, por exemplo); plataforma de execução (*Windows* 2000, *Windows* NT...); padrões de modularização (nome de arquivos executáveis...); robustez, integridade e segurança (condições adversas sob as quais a aplicação deve manter-se em operação; integridade dos arquivos de dados em panes elétricas...; política de classificação de usuários, com especificação de restrições de acesso a dados e funcionalidades...), performance (tempos de resposta); manutenção (restrições de tempo e esforço para proceder alterações emergenciais referentes a modificação de legislação...).

6.1.2 Documento do Plano de Testes

Este documento é um dos documentos pertencentes à estrutura de documentos textuais criada no ambiente SEA. A finalidade de um plano de teste é estabelecer os casos de teste que devem ser testados, além de possuir informações que identificam um plano de teste, como seção inicial (identificador do documento), histórico do documento e introdução.

Neste documento, alguns conceitos são reutilizados pelo documento de especificação de requisitos, porém serão descritos no mesmo. A seguir são descritos os conceitos e seus respectivos atributos referenciados por este documento (modelo).

O documento plano de teste, assim como o documento de especificação de requisitos, foi organizado nas seções descritas a seguir. Cada seção é definida como um conceito de OCEAN:

- **Identificador do documento**;
- **Histórico do documento**;
- **Introdução**;
- **Casos de Teste**.

O diagrama da figura 6.2 representa a organização do documento de plano de teste e as seções que fazem parte do mesmo.

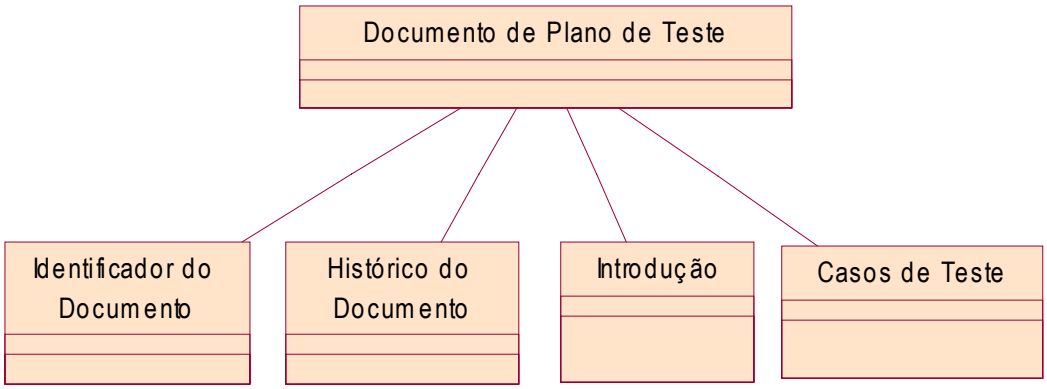


Figura 6.2 - Organização do Documento de Plano de Teste

6.1.2.1 Conceito IdentificadorDoc

Este conceito é representado por um identificador do documento e possui as mesma finalidade que um identificador do documento num documento de especificação de requisitos, conforme descrito anteriormente. O conceito IdentificadorDoc possui os seguintes atributos:

- **Identificação da Empresa**;
- **Projeto:** identificação do projeto;
- **Aplicação - Parte do Projeto:** Identificação do *software* especificado;
- **Tipo de Documento (padrão da empresa)**;
- **Identificador do Documento (padrão da empresa)**;
- **Versão do Documento**;
- **Data da Conclusão do Documento**.

6.1.2.2 Conceito HistoricoDocumento

Este conceito agrega uma lista de item de históricos, outro conceito que possui seus atributos específicos, assim como no documento de especificação e possui a mesma finalidade que no documento citado. A coleção do HistoricoDocumento é apresentada automaticamente, assim que o conceito ItemHistorico for referenciado para preencher novo item.

6.1.2.2.1 Conceito ItemHistorico

Este conceito agrega atributos que formam um item de histórico. Verificar descrição do conceito no documento de especificação de requisitos. O conceito ItemHistorico possui os seguintes atributos:

- **Versão do Documento**;
- **Autor (es) do Documento**;
- **Data do Documento**;
- **Ação praticada**;

- **Esforço (em horas, especificado por autor)**.

6.1.2.3 Conceito Introducao

Este conceito contém informações necessárias à introdução de um documento de plano de testes (verificar descrição no documento de especificação de requisitos) e agrega os seguintes atributos:

- **Objetivo:** definição em linhas gerais dos objetivos dos testes a serem aplicados e identificação;
- **Descrição:** descrição textual, se necessário;
- **Informações Gerais:** informações sobre os testes, se necessário;
- **Localização do documento:** referência do armazenamento do presente documento plano de teste.

6.1.2.4 Conceito CasosTeste

Este conceito contém uma coleção de caso de teste, outro conceito que possui seus atributos específicos. A coleção de caso de teste é definida por meio do conceito CasoTeste.

A coleção do CasosTeste é gerada automaticamente, assim que o conceito CasoTeste for referenciado para preencher novo item caso de teste.

6.1.2.4.1 Conceito CasoTeste

O conceito CasoTeste é referenciado pelo conceito CasosTeste. Este conceito possui informações referentes à identificação e descrição de casos de teste que serão definidos no documento de plano de teste. Este conceito possui os seguintes atributos a serem informados:

- **Identificação Caso Teste:** identificação do caso de teste;
- **Descrição Caso de Teste:** descrição do caso de teste;

- **Descrição do Procedimento de Teste**;
- **Dados de Entrada:** dados fornecidos;
- **Resultado Esperado:** dados esperados com a aplicação de teste.

6.1.3 Documento Formulário de Aplicação de Teste

Este documento também foi inserido à estrutura de documentos textuais do ambiente SEA. Ele complementa um plano de teste, pois o documento formulário de aplicação de teste possui a finalidade de registrar a realização de testes baseados nos casos de teste definidos no documento de plano de teste.

A seguir, são apresentados os conceitos referenciados pelo formulário de aplicação de teste e os respectivos atributos pertencentes a cada conceito, isto é, informações que devem ser fornecidas por usuários do ambiente SEA.

O formulário de aplicação de teste também definido no estudo de caso foi organizado nas seções seguintes. Cada seção é definida como um conceito de OCEAN.

- **Identificação**;
- **Agendamento**;
- **Colaboradores**;
- **Resultado de Aplicação de Teste**.

O diagrama representado pela figura 6.3 apresenta tal organização.

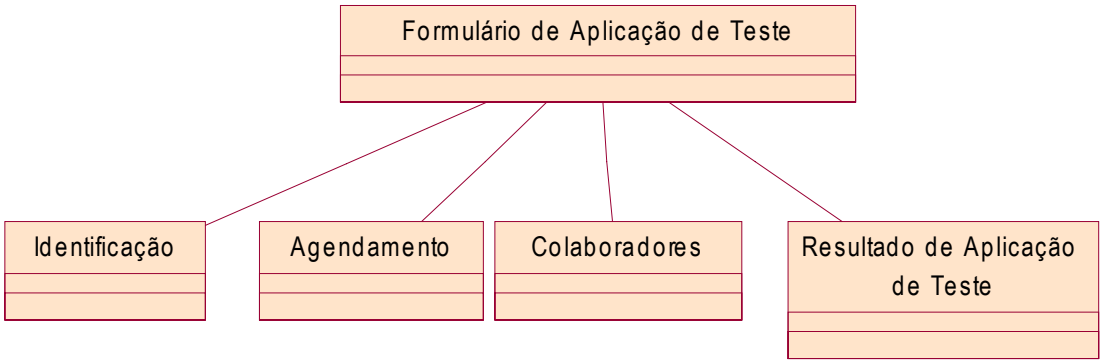


Figura 6.3 - Organização do Formulário de Aplicação de Teste

6.1.3.1 Conceito Identificação

O conceito Identificação possui duas instâncias: uma instância para teste e outra para produto testado. Este conceito é responsável por identificar informações referentes ao produto que está sendo tratado e informações referentes ao teste que está sendo aplicado.

Neste conceito, deve ser informado primeiramente o tipo de identificação a ser procedida, ou seja, num formulário de aplicação de teste, definido pelo próprio desenvolvedor da estrutura de documentos textuais.

Os atributos deste conceito, ou seja, as informações a serem definidas são as mesmas, tanto para identificação do teste quanto para identificação do produto testado. O que muda são as informações referentes a cada atributo. A seguir são apresentados os atributos que o conceito corrente agrega, ou seja, as instâncias do conceito.

Instância Identificação do Teste:

- **Tipo:** tipo do teste a ser aplicado (validação, por exemplo);
- **Identificador:** deve ser o mesmo identificador de documento (capa) do plano de teste;
- **Versão:** deve ser a mesma do identificador de documento (capa) do plano de teste;
- **Localização:** localização do documento - deve-se verificar na base de dados do projeto.

Instância Identificação do Produto Testado:

- **Tipo:** tipo do produto testado;
- **Identificador:** Módulo de habilitação;
- **Versão**;
- **Localização**.

6.1.3.2 Conceito Agendamento

Este conceito é responsável por fornecer as informações para agendar uma determinada atividade, neste caso, agendar uma aplicação de teste. Os atributos agregados por este conceito referentes às informações presentes num agendamento são:

- **Data:** data da aplicação do teste;
- **Hora:** hora da aplicação do teste;
- **Local:** local a ser aplicado o teste;
- **Duração Prevista (h):** duração prevista para aplicação do teste;
- **Duração Real (h):** duração real da aplicação do teste.

6.1.3.3 Conceito Colaboradores

O conceito colaboradores é responsável por apresentar as pessoas envolvidas numa aplicação de teste, nesse caso. As informações dos colaborares são obtidas através do conceito PessoaEnvolvida, pois este conceito possui informações a serem definidas em relação às pessoas que colaboram para aplicação do teste.

Este conceito possui uma coleção de colaboradores identificada pela função do colaborador. O conceito corrente referencia o conceito PessoaEnvolvida responsável por fornecer as informações a serem procedidas por usuários do ambiente SEA referente às funções, nomes e esforços dos colaboradores em determinada aplicação de teste.

6.1.3.3.1 Conceito PessoaEnvolvida

Este conceito é referenciado pelo conceito Colaboradores como descrito no item 6.1.3.3. O conceito PessoaEnvolvida é responsável por permitir que informações sobre as pessoas envolvidas na aplicação do teste sejam definidas. Os atributos deste conceito a serem informados são os seguintes:

- **Função do Colaborador:** função da pessoa envolvida na aplicação do teste - no caso dos colaboradores da aplicação do teste, a função de quantos forem os colaboradores, será de aplicador do teste, por exemplo, aplicador 1, aplicador 2...;
- **Nome do Colaborador:** nome da pessoa envolvida na aplicação do teste, ou seja, do aplicador;
- **Esforço do Colaborador (h):** esforço de aplicação do teste por colaborador (aplicador).

6.1.3.4 Conceito ResultadoAplicacaoTeste

O conceito ResultadoAplicacaoTeste é responsável por apresentar as informações necessárias de uma aplicação de teste. A Aplicação de teste é em relação aos casos de testes definidos no documento de plano de teste.

Este conceito possui um conjunto de resultados de aplicação de teste em que cada resultado é identificado nessa lista pelo identificador do caso de teste e sua descrição herdada do documento do plano de teste através do conceito casos teste.

Ao selecionar um caso de teste da lista do conceito corrente, é referenciado outro conceito, denominado RegistroTeste, que possui as informações referentes a uma aplicação de caso de teste. Neste conceito o único atributo gerado automaticamente é a lista.

6.1.3.4.1 Conceito RegistroTeste

O conceito RegistroTeste possui a finalidade de fornecer informações a serem definidas em relação à aplicação do teste nos casos de teste. Este conceito é referenciado pelo conceito ResultadoAplicacaoTeste, pois possui os dados a serem informados no documento formulário de aplicação de teste. Os atributos pertencentes a este conceito que devem ser informados são:

- **Identificador do Caso Teste:** este conceito é herdado automaticamente do documento plano de teste através do conceito CasoTeste;

- **Descrição Caso Teste:** este conceito é herdado automaticamente do documento plano de teste através do conceito caso teste (atributo descrição CasoTeste);
- **Resultado**: resultado da aplicação do caso de teste, por exemplo, se está "ok" ou se houve falha;
- **Observação:** caso tenha falhado o caso de teste testado, pode ser descrita a falha;
- **OK do Autor:** ok do Autor na aplicação do teste (para validação da aplicação do teste);
- **OK do Coordenador:** ok do coordenador na aplicação do teste (para validação da aplicação do teste).

6.1.4 Documento Formulário de Inspeção

Este é outro documento pertencente à estrutura de documentos textuais do ambiente SEA. O formulário de inspeção é responsável por disponibilizar informações que precisam ser estabelecidas durante uma inspeção.

A seguir, são apresentados os conceitos referenciados pelo formulário de inspeção e os seus atributos pertencentes a cada conceito, isto é, dados que devem ser informados por usuários do ambiente SEA.

Alguns conceitos referenciados pelo formulário de inspeção também são referenciados pelo formulário de aplicação de teste, descritos no de inspeção.

O formulário de inspeção definido no estudo de foi organizado nas seguintes seções, cada uma definida como um conceito de OCEAN.

- **Identificação**;
- **Agendamento**;
- **Colaboradores**;
- **Inspeção**.

A figura 6.4 apresenta através do diagrama a forma de como as seções estão organizadas no formulário de inspeção.

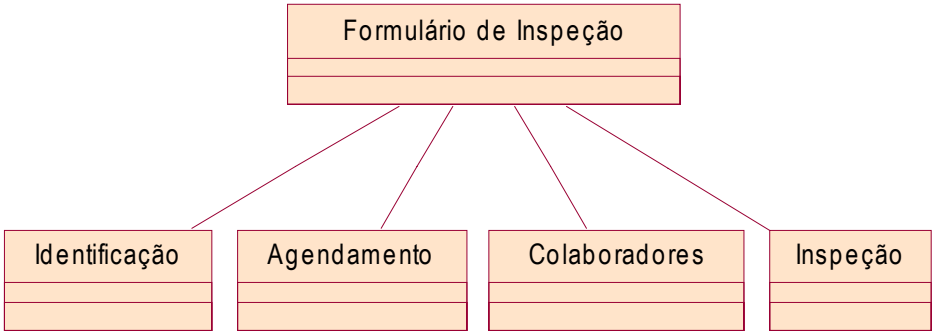


Figura 6.4 - Organização do Formulário de Inspeção

6.1.4.1 Conceito Identificação

Neste conceito, deve ser informado primeiramente o tipo de identificação a ser procedida, ou seja, no formulário de inspeção, definido pelo próprio desenvolvedor da estrutura de documentos textuais, foi estabelecido que uma identificação deve ter uma instância do tipo identificação do produto.

O conceito Identificação também é referenciado pelo FormularioAplicacaoTeste, com os mesmos atributos mas tipos de identificação diferentes, assim como as informações preenchidas em cada atributo.

Os atributos deste conceito, ou seja, as informações a serem definidas para identificação do produto são as seguintes:

Identificação do Produto:

- **Tipo:** projeto;
- **Identificador:** identificação do produto, por exemplo, requisitos funcionais do XXX;
- **Versão:** versão do documento de inspeção;

- **Localização:** localização do documento de inspeção, ou seja, onde está armazenado o documento.

6.1.4.2 Conceito Agendamento

Este conceito é responsável por fornecer as informações para agendar uma determinada atividade, neste caso, uma inspeção. O conceito Agendamento também é reutilizado pelo documento Formulário de Aplicação de Teste. Os atributos agregados por este conceito referentes às informações presentes num agendamento são:

- **Data:** data da inspeção;
- **Hora:** hora da inspeção;
- **Local:** local a ser inspeção;
- **Duração Prevista (h):** duração prevista para inspeção;
- **Duração Real (h):** duração real da inspeção.

6.1.4.3 Conceito Colaboradores

Este conceito possui um conjunto de colaboradores identificados pela função do colaborador assim como o documento formulário de aplicação de teste. O conceito corrente referencia o conceito PessoaEnvolvida responsável por fornecer as informações a serem procedidas por usuários do ambiente SEA referente às funções, nomes e esforços dos colaboradores em determinada inspeção.

6.1.4.3.1 Conceito PessoaEnvolvida

Este conceito é referenciado pelo conceito Colaboradores como descrito no item 6.1.4.3. Verificar descrição no formulário de aplicação de teste. Os atributos agregados por este conceito referente às informações necessárias são:

- **Função do Colaborador:** função da pessoa envolvida na aplicação do teste - no caso dos colaboradores da inspeção, deve haver pelo menos um autor, pelo menos um coordenador, e pelo menos inspetor;
- **Nome do Colaborador:** nome da pessoa envolvida na aplicação do teste, de acordo com a função definida;
- **Esforço do Colaborador (h):** esforço de aplicação do teste por colaborador.

6.1.4.4 Conceito Inspecao

O conceito Inspeção (sem acentuação, devido estar tratando o conceito, ou seja, o nome definido na implementação) é responsável por apresentar as informações referentes a uma inspeção e o registro de erros encontrado durante a mesma.

Este conceito possui uma coleção de registros de erros, na qual a chave da lista é identificada pela ordem de um registro de erros herdado do conceito Erro, referenciado pelo conceito Inspecao, por meio do atributo ordem do conceito Erro.

Neste conceito, o único atributo gerado automaticamente é a coleção, desde que tenha referenciado pelo menos uma vez o conceito Inspecao.

6.1.4.4.1 Conceito Erro

O conceito Erro possui as informações relacionadas a um registro de inspeção, bem como a descrição do erro identificado durante o registro. Este conceito é referenciado pelo conceito Inspeção e agrega os seguintes atributos referentes às informações que devem ser definidas:

- **Ordem:** campo automático, incrementando toda vez que for registrado um erro;
- **Localização no Documento:** localização do documento, ou seja, no histórico do documento; na página 3 da seção 2...;
- **Gravidade:** pode ser estabelecido pela empresa o nível de gravidade;
- **Descrição do Erro:** descrição do erro encontrado durante a inspeção;

- **Referência *Bugzilla***;
- **Esforço (h):** esforço durante toda a inspeção;
- **OK do Autor:** ok do autor na inspeção;
- **OK do Coordenador:** ok do coordenador na inspeção.

6.2 Regras de Consistência dos Documentos Textuais Estruturados

Cada documento possui suas regras de consistência. As regras de consistência de um documento são determinadas pelas regras impostas a cada conceito referenciado por determinado documento e pelas regras impostas na criação de modelos (documentos).

As regras de consistência foram definidas de acordo com as informações identificadas como essenciais nos documentos tratados no ambiente SEA.

Cada modelo, ou seja, documento, será consistente se todas as regras descritas, forem obedecidas. Se, pelo menos, uma das regras de consistência não for seguida, o documento ou a especificação estará inconsistente.

O resultado da verificação de consistência é armazenado num documento texto, conferido por meio da análise das regras de consistência do documento corrente, regras implementadas pelas ferramentas de análise de consistência que serão descritas no capítulo 7.

Ressalta-se a vantagem em utilizar um documento com essa estrutura, pois é possível realizar a verificação de consistência nos documentos criados, o que não ocorre num documento do *Word*, por exemplo.

A seguir são descritas as regras de consistência para os documentos tratados na Estrutura de Documentos Textuais implantada no ambiente SEA.

6.2.1 Regras de Consistência do Documento de Especificação de Requisitos

Cada modelo possui suas próprias regras, as quais devem ser obedecidas para manter a consistência das informações. A seguir são descritas as regras de consistência do documento de especificação de requisitos.

6.2.1.1 Regras para Criação do Documento de Especificação de Requisitos

Para criação de um documento de especificação de requisitos há somente uma restrição: informar um nome para o documento de especificação de requisitos, obrigatoriamente.

Cada documento de plano de deve ter obrigatoriamente um(a) e somente um(a):

- **Identificador do Documento**;
- **Histórico do Documento**;
- **Introdução**;
- **Visão Geral do Sistema**;
- **Conjunto de Requisitos**.

6.2.1.2 Regras de Consistência dos Conceitos do Documento de Especificação de Requisitos

Cada conceito possui suas próprias regras para serem analisadas em conjunto pela ferramenta de análise do modelo corrente. As regras de consistência dos conceitos pertencentes a este modelo são descritas a seguir.

❖ **Conceito IdentificadorDoc**

O conceito IdentificadorDoc significa identificador do documento, ou seja, uma capa ou página inicial. Este conceito possui algumas informações obrigatórias, as quais fazem parte das regras de consistência deste conceito. A seguir, são descritas as regras de consistência do conceito identificador do documento.

O Identificador do Documento deve ter obrigatoriamente os campos:

- **Identificação da Empresa:** campo do tipo *String*; campo não vazio;
- **Identificação do Projeto:** campo do tipo *String*; campo não vazio;
- **Parte do Projeto:** campo do tipo *String*; campo não vazio;
- **Identificador do Documento:** campo do tipo *String*; campo não vazio;
- **Versão:** campo do tipo *Number*; campo não vazio;
    - Este campo não foi definido automaticamente, por exemplo, 1, 2 ,3, porque o usuário do ambiente SEA pode querer especificar 1.0, 1.1 e assim por diante;
- **Data:** campo do tipo *Date*, ou seja, deve ser informado da seguinte forma: mês/dia/ano; em todos os conceitos que possui o atributo data; informação obrigatória;
    - Não pode haver uma data com o ano inferior a 2003;

❖ **Conceito HistoricoDocumento**

Este conceito significa histórico do documento. O conceito HistoricoDocumento referencia o conceito ItemHistorico, herdando as informações contidas nele. Portanto, um histórico do documento pode ter um ou vários itens de histórico (ItemHistorico).

Para manter a consistência das informações do conceito descrito, alguns campos são obrigatórios que sejam informados. As regras de consistência desse conceito são as seguintes:

- **Versão:** campo do tipo *Number*; campo não vazio;
    - O campo versão deve ser informado para proceder às informações na lista do Histórico do Documento;
    - A última versão a ser criada num Item Histórico deve ser superior a uma versão existente; e diferente de alguma que já exista;
- **Autor(es):** campo do tipo *String*; campo não vazio;
    - Deve haver pelo menos um Autor;
- **Data:** campo do tipo *Date*; campo não vazio;
    - Deve ser informado mês/dia/ano;
    - Não pode haver uma data com o ano inferior a 2003;

- Não pode haver uma data igual a uma existente;
- A última data a ser definida deve ser superior a uma que já exista;

- **Ação:** campo do tipo *String*; campo não vazio;
- **Esforço:** campo do tipo *Number*; campo não vazio;
  - Deve ser definida em número (s);

❖ **Conceito Introducao**

Este conceito significa introdução, bem como representa o nome definido no conceito. Para manter a consistência das informações do conceito descrito, alguns campos obrigatoriamente devem ser informados. Neste conceito, existem quatro campos que devem ser informados, mas apenas dois não podem ser vazios. A seguir são descritas as regras de consistência do conceito corrente.

- **Objetivo:** campo tipo *Text*; campo não vazio;
  - Deve haver o objetivo do desenvolvimento obrigatoriamente;
- **Descrição:** campo tipo *Text*; campo não vazio;
  - Deve haver a descrição obrigatoriamente;
- **Informações Gerais:** campo tipo *Text*; campo não obrigatório;
- **Localização:** campo tipo *Text*; campo não obrigatório.

❖ **Conceito VisaoGeralSistema**

Este conceito representa a visão geral do sistema. Para manter a consistência das informações do conceito descrito, alguns dos campos devem obrigatoriamente ser informados. A seguir são descritas as regras de consistência do conceito VisaoGeralSistema.

- **Arquitetura do Sistema**: campo tipo *Text*; campo não vazio;
  - Deve haver a descrição da arquitetura do sistema;
- **Arquitetura da Aplicação:** campo tipo *Text*; campo não obrigatório;
- **Premissas de Desenvolvimento:** campo tipo *Text*; campo não vazio;
  - Deve haver a descrição das premissas de desenvolvimento;

❖ **Conceito ConjuntoRequisitos**

Este conceito representa o conjunto dos requisitos envolvidos num projeto. O conceito ConjuntoRequisitos referencia o conceito Requisito, herdando as informações contidas nele. Portanto um ConjuntoRequisitos pode ter um ou vários requisitos (Requisito) e deve haver pelo menos um requisito funcional.

Para manter a consistência das informações do conceito descrito, é obrigatório que alguns campos sejam informados. As regras de consistência desse conceito são as seguintes:

- **Identificação do Requisito:** campo tipo *String*; campo não vazio;
    - O documento de especificação de requisitos deve ter pelo menos um Requisito Funcional;
    - Pode ter um ou vários requisitos de interface;
    - Pode ter uma ou várias restrições de projeto (requisitos não-funcionais);
- **Descrição do Requisito:** campo tipo *Text*; campo não vazio;
    - Deve haver a descrição de cada requisito criado;

6.2.2 Regras de Consistência do Documento de Plano de Testes

A seguir são descritas as regras de consistência do documento de plano de teste.

6.2.2.1 Regras para Criação do Documento de Plano de Testes

Para criação de um documento de plano de testes há somente uma restrição, ou seja, não há nenhuma regra imposta. Cada documento de especificação de requisitos deve ter obrigatoriamente um(a) e somente um(a):

- **Identificador do Documento**;
- **Histórico do Documento**;
- **Introdução**;
- **Casos de Teste**.

6.2.2.2 Regras de Consistência dos Conceitos do Documento de Plano de Testes

As regras de consistência dos conceitos pertencentes a este modelo são descritas a seguir.

❖ **Conceito IdentificadorDoc**

Este conceito é o mesmo referenciado pelo documento de especificação de requisitos e assim como nesse documento, o documento plano de testes possui as mesmas informações obrigatórias relativas ao conceito em questão, as quais fazem parte das suas regras de consistência. A seguir, são descritas as regras de consistência do conceito identificador do documento.

O Identificador do Documento deve ter obrigatoriamente os campos:

- **Identificação da Empresa:** campo do tipo *String*; campo não vazio;
- **Identificação do Projeto:** campo do tipo *String*; campo não vazio;
- **Parte do Projeto:** campo do tipo *String*; campo não vazio;
- **Identificador do Documento:** campo do tipo *String*; campo não vazio;
- **Versão:** campo do tipo *Number*; campo não vazio;
    - Este campo não foi definido automaticamente, por exemplo, 1, 2 ,3, porque o usuário do ambiente SEA pode querer especificar 1.0, 1.1 e assim por diante;
- **Data:** campo do tipo *Date*, ou seja, deve ser informado da seguinte forma: mês/dia/ano; em todos os conceitos que possui o atributo data; campo não vazio;
    - Não pode haver uma data com o ano inferior a 2003.

❖ **Conceito HistoricoDocumento**

Este conceito é referenciado não apenas pelo documento de plano de testes como documento de especificação de requisitos e significa histórico do documento. O conceito HistoricoDocumento referencia o conceito ItemHistorico, herdando as informações contidas nele e um histórico do documento pode ter um ou vários itens de histórico (ItemHistorico), como mencionado anteriormente, e deve haver pelo menos um.

Para manter a consistência das informações do conceito descrito, é obrigatório que alguns campos sejam informados. As regras de consistência desse conceito são as seguintes:

- **Versão:** campo do tipo *Number*; campo não vazio;
    - O campo versão deve ser informado para proceder às informações na lista do Histórico do Documento;
    - A última versão a ser criada num Item Histórico deve ser superior a uma versão existente; e diferente de alguma que já exista;
- **Autor(es):** campo do tipo *String*; campo não vazio;
    - Deve haver pelo menos um Autor;
- **Data:** campo do tipo *Date*; campo não vazio;
    - Deve ser informado mês/dia/ano;
    - Não pode haver uma data com o ano inferior a 2003;
    - Não pode haver uma data igual a uma existente;
    - A última data a ser definida deve ser superior a uma que já exista;
- **Ação:** campo do tipo *String*; campo não vazio;
- **Esforço:** campo do tipo *Number*; campo não vazio;
    - Deve ser definida em número (s);

❖ **Conceito Introducao**

Este conceito também é referenciado pelo documento de especificação de requisitos e utiliza as mesmas regras tanto para o documento de especificação quanto para o documento de plano de testes.

Neste conceito, existem quatro campos que devem ser informados, mas apenas dois não podem ser vazios. A seguir são descritas as regras de consistência do conceito corrente.

- **Objetivo:** campo tipo *Text*; campo não vazio;
    - Deve haver o objetivo do desenvolvimento;
- **Descrição:** campo tipo *Text*; campo não obrigatório;
- **Informações Gerais:** campo tipo *Text*; campo não obrigatório;
- **Localização:** campo tipo *Text*; campo não vazio.
    - Deve haver a localização do documento;

❖ **Conceito CasosTeste**

Este conceito representa Casos de Teste. O conceito CasosTeste referencia o conceito CasoTeste, herdando as informações contidas nele. Contudo, um conceito CasosTeste pode ter um ou vários conceitos CasoTeste, mas deve ter pelo menos um.

Para manter a consistência das informações do conceito descrito, é obrigatório que alguns campos sejam informados. As regras de consistência desse conceito são as seguintes:

- **Identificador do Caso de Teste**: campo tipo *Number ready only*, ou seja, somente leitura; Inserido ordenado automaticamente. Por exemplo: 1, 2, 3...;
- **Descrição do Caso de Teste**: campo tipo *String*; campo não vazio;
- **Dados de Entrada**: campo tipo *Text*; campo não vazio;
- **Resultado Esperado:** campo tipo *Text*; campo não vazio;

Caso uma dessas regras não esteja sendo obedecida, é informada a inconsistência de determinado conceito, e, conseqüentemente, do documento corrente.

6.2.3 Regras de Consistência do Formulário de Aplicação de Teste

São descritas, a seguir, as regras de consistência do documento formulário de aplicação de teste.

6.2.3.1 Regras para Criação do Formulário de Aplicação de Teste

Para criação de um formulário de aplicação de teste, há algumas restrições obrigatórias como:

- Primeiramente deve haver um documento de plano de teste criado;
- No documento de plano de teste, deve ter ao menos um conceito casos de teste criado;

- O formulário de aplicação de teste a ser criado deve estar associado a um conceito CasosTeste;
    - Caso os três requisitos anteriormente descritos não sejam obedecidos, é informado que o formulário de aplicação de teste deve ter um conceito CasosTeste associado a ele e, por isso, o formulário não poderá ser criado;
- O formulário de aplicação de teste deve possuir um nome associado a ele;
- Após ter obedecido às regras descritas acima, o formulário somente será criado caso cumpra essas restrições. Ao ser criado, um formulário de aplicação de teste deve possuir obrigatoriamente:
    - **Identificação:** no formulário de aplicação deve haver duas identificações: uma identificação do teste e uma identificação do produto testado. Caso se tente, porém, criar duas identificações do teste ou duas identificações do produto testado, ou ainda uma terceira, o conceito (identificação) será inconsistente. Por isso, à medida que este conceito seja criado deve-se informar o nome deste tipo de conceito, ou se já, uma identificação do teste e outra do produto testado;
    - **Agendamento:** deve haver somente um agendamento para cada documento;
    - **Colaboradores:** deve haver somente um conceito colaboradores;
    - **Resultado de Aplicação do Teste:** deve haver somente um conceito resultado de aplicação de teste.

6.2.3.2 Regras de Consistência dos Conceitos do Formulário de Aplicação de Teste

As regras de consistência dos conceitos pertencentes a este modelo são descritas a seguir.

❖ **Conceito Identificacao**

Este conceito representa a identificação de produto e teste num formulário de aplicação de teste. Nele, informações necessárias ao produto e ao teste devem ser definidas. Pois, como relatado anteriormente, o formulário de aplicação de teste deve ter uma identificação do teste e uma identificação do produto testado obrigatoriamente. Os dois tipos de identificação possuem os mesmos atributos a serem informados num contexto diferente e é

o único conceito que, ao ser criado, deve-se informar o nome: teste e outro, para produto testado.

O conceito Identificacao é referenciado pelo formulário de aplicação de teste e possui as seguintes regras de consistência:

- **Identificador**: campo tipo *String*; campo não vazio; *read only*, ou seja, somente para leitura (apenas para identificação do teste, do produto testado, não);
    - Para identificação do teste, este campo deve ser herdado do documento de plano de testes;
- **Versão:** campo tipo *String*; campo não vazio; *read only*, ou seja, somente para leitura;
    - Para identificação do teste, a versão deve ser herdada do documento de plano de testes;
- **Localização:** campo tipo *String*; campo não vazio;

Para a identificação do produto testado, todas as informações devem ser relatadas, não havendo nenhuma particularidade em relação ao produto testado.

❖ **Conceito Agendamento**

Este conceito é responsável pelo agendamento de uma aplicação de teste. O conceito agendamento é referenciado pelo formulário de aplicação de teste.

Neste conceito, existem campos que devem ser informados e não podem ser vazios. A seguir são descritas as regras de consistência do conceito corrente.

- **Data**: campo tipo *Date*; campo não vazio;
    - Deve ser informado mês/dia/ano;
    - Não pode haver uma data com o ano inferior a 2003;
- **Hora**: campo tipo Time; campo não vazio;
    - A hora é definida como, por exemplo, 00:00:00;
- **Duração Prevista (h):** campo tipo *Number*; campo não vazio;
    - A duração prevista deve ser definida em horas, ou seja, a unidade hora;
- **Duração Real (h):** campo tipo *Number*; campo não vazio;
    - A duração real deve ser definida em horas, ou seja, na unidade hora.

❖ **Conceito Colaboradores**

Este conceito representa as pessoas envolvidas na aplicação de teste.

O conceito Colaboradores referencia o conceito PessoaEnvolvida, herdando as informações contidas nele. Portanto, um conceito colaboradores pode ter um ou vários colaboradores (PessoaEnvolvida).

Para manter a consistência das informações do conceito descrito, é obrigatório que alguns campos sejam informados. As regras de consistência desse conceito são as seguintes:

- **Nome do Colaborador:** campo do tipo *String*; campo não vazio;
    - Deve haver pelo menos um colaborador;
- **Função do Colaborador:** campo do tipo *String*; campo não vazio;
    - Normalmente, a função é: aplicador;
- **Esforço do Colaborador:** campo do tipo *Number*; campo não vazio;
    - O esforço deve ser definido em horas, ou seja, na unidade hora;
    - Cada colaborador possui seu próprio esforço;

❖ **Conceito ResultadoAplicacaoTeste**

Este conceito representa o resultado da aplicação de testes, ou seja, a aplicação de cada teste definido no documento de plano de teste.

O conceito ResultadoAplicacaoTeste referencia o conceito RegistroTeste, herdando as informações contidas nele. Contudo, um conceito resultado da aplicação de testes pode ter um ou vários resultados de aplicação de teste. Isso depende do número de casos de teste definido no documento de plano de testes.

Ressalta-se que este conceito só poderá ser procedido em relação às informações, caso tenha sido criado no documento de plano de teste, pelo menos um CasosTeste. Caso contrário, não poderá ser executado o procedimento de aplicação de teste, porém o conceito ResultadoAplicacaoTeste até poderá existir, mas as informações necessárias referentes este

conceito, não existirão, pois é através de uma lista de casos de teste que as informações relativas à aplicação do teste são definidas.

Para manter a consistência das informações do conceito descrito, obrigatoriamente alguns campos devem ser informados.

- **Identificador do Caso de Teste:** campo tipo *Number*; campo não vazio, herdado automaticamente do conceito CasosTeste do documento de plano de testes; como descrito no documento de plano de teste, este campo é inserido automaticamente;
- **Descrição do Caso de Teste:** campo tipo *String*; campo não vazio, herdado automaticamente do conceito CasosTeste do documento de plano de testes referente ao identificador do caso de teste;
- **Resultado:** campo tipo *String*; campo não vazio;
    - Deve ser informado obrigatoriamente o resultado do teste, por exemplo, se houve falha ou se está tudo ok em relação ao teste aplicado.
- **Observação:** campo tipo *String*; campo não vazio;
    - No campo observação, deve ser informado que tipo de falha ocorreu em relação ao teste aplicado ou, caso não tenha ocorrido nenhuma falha, deve haver uma informação de que não houve falha simplesmente representar por um "----";
- **Ok do Autor:** campo tipo *String*; campo não vazio;
    - Neste caso deve haver um ok ou um "x" para representar que o autor da teste aplicado concluiu a sua aplicação;
- **OK do Coordenador:** campo tipo *String*; campo não vazio;
    - Neste caso também deve ser definido um ok ou um "x" para representar que o coordenador está informado da aplicação do teste.

Caso uma dessas regras não esteja sendo obedecida, é informada a inconsistência de determinado conceito, e, conseqüentemente, do documento corrente.

6.2.4 Regras de Consistência do Formulário de Inspeção

A seguir são descritas as regras de consistência do documento formulário de inspeção.

6.2.4.1 Regras para Criação do Formulário de Inspeção

Para criação de um formulário de inspeção, há algumas restrições obrigatórias como:

- Primeiramente deve haver um nome associado ao formulário de inspeção;
- Deve haver um *link* semântico associado ao formulário de inspeção;
    - O *link* semântico pode estar associado a qualquer documento, especificação, *framework*;
    - Lembrando que o documento formulário de inspeção até pode ser criado e possuir todos os seus conceitos e atributos seguindo as regras de consistência dos conceitos, mas, caso não tenha um *link* semântico associado a ele, o documento estará inconsistente, porém, não pode haver só o *link* associado sem as outras regras de consistência (regras dos conceitos), isto é, uma regra complementa outra;
- Um formulário de inspeção deve possuir obrigatoriamente:
    - **Identificação:** no formulário de inspeção, diferente do formulário de aplicação de teste, deve haver somente uma identificação: a do produto; os atributos deste conceito são os mesmos, tanto para identificação do formulário de aplicação de teste quanto para o de inspeção; ao criar este conceito no formulário de inspeção, deve informar, "identificação do produto".
    - **Agendamento:** deve haver somente um agendamento para cada documento;
    - **Colaboradores:** deve haver somente um conceito colaboradores;
    - **Inspeção:** deve haver somente um conceito Inspecao.

6.2.4.2 Regras de Consistência dos Conceitos do Formulário de Inspeção

As regras de consistência dos conceitos pertencentes a este modelo são descritas logo abaixo.

❖ **Conceito Identificacao**

Este conceito representa a identificação do produto, no caso do formulário de inspeção.

Como relatado anteriormente, o formulário de inspeção deve ter somente uma identificação do produto. É o único conceito que, ao ser criado, deve-se informar o nome: neste caso, identificação do produto.

Este conceito é referenciado pelo formulário de inspeção e possui as seguintes regras de consistência:

- **Identificador:** campo tipo *String*; campo não vazio; neste caso, o projeto a ser inspecionado;
- **Versão:** campo tipo *String*; campo não vazio;
- **Localização:** campo tipo *String*; campo não vazio.

❖ **Conceito Agendamento**

Este conceito é responsável pelo agendamento de uma inspeção. O conceito agendamento é referenciado também pelo formulário de aplicação de teste.

Neste conceito, existem campos que devem ser informados e não podem ser vazios. A seguir são descritas as regras de consistência do conceito corrente. As regras são as mesmas para o formulário de inspeção e para o de aplicação de teste.

- **Data:** campo tipo *Date*; campo não vazio;
    - Deve ser informado mês/dia/ano;
    - Não pode haver uma data com o ano inferior a 2003;
- **Hora:** campo tipo *Time*; campo não vazio;
    - A hora é definida como, por exemplo, 00:00:00;
- **Duração Prevista (h):** campo tipo *Number*; campo não vazio;
    - A duração prevista deve ser definida em horas, ou seja, a unidade hora;
- **Duração Real (h):** campo tipo *Number*; campo não vazio;
    - A duração real deve ser definida em horas, ou seja, na unidade hora;

❖ **Conceito Colaboradores**

Este conceito representa as pessoas envolvidas numa inspeção. Este conceito referencia o conceito PessoaEnvolvida, herdando as informações contidas nele. Portanto, um conceito colaboradores pode ter um ou vários colaboradores (PessoaEnvolvida).

Para manter a consistência das informações do conceito descrito, alguns campos são obrigatórios. No formulário de inspeção, as regras de consistência são diferentes do que as regras para o formulário de aplicação de teste, porém, os atributos são os mesmos. As regras de consistência desse conceito são as seguintes:

- **Nome do Colaborador:** campo do tipo *String*; campo não vazio;
    - Deve haver pelo menos um colaborador;
- **Função do Colaborador:** campo do tipo *String*; campo não vazio;
    - No formulário de inspeção, pode haver: Autor(es), Coordenador, Inspetor(es);
    - Deve haver pelo menos um autor, obrigatoriamente;
    - Deve haver somente um coordenador, obrigatoriamente;
    - Deve haver pelo menos um inspetor, obrigatoriamente;
- **Esforço do Colaborador:** campo do tipo *Number*; campo não vazio;
    - O esforço deve ser definido em horas, ou seja, na unidade hora;
    - Cada colaborador possui seu próprio esforço;

❖ **Conceito Inspecao**

Este conceito significa o resultado de um registro de inspeção, ou seja, o registro de inspeção realizado no documento que foi associado ao formulário através do *link* semântico.

O conceito RegistroErro referencia o conceito Erro, herdando as informações contidas nele. Contudo, um conceito registro de erro pode ter um ou vários erros. Isto depende do número de casos de teste definido no documento de plano de teste.

Para manter a consistência das informações do conceito descrito, alguns campos são obrigatórios que sejam informados. A seguir são descritas as regras de consistência do conceito corrente para o documento formulário de inspeção.

- **Ordem:** campo tipo *Number*; campo não vazio; criado automaticamente de forma ordenada, por exemplo, 1, 2, 3...;
- **Localização do Documento:** campo tipo *String*; campo não vazio;
- **Gravidade:** campo tipo *String*; campo não é obrigatório;
- **Descrição do Erro:** campo tipo *String*; campo não vazio;
    - Caso não tenha descrição de erro, informar que não teve erro no documento inspecionado;
- **Referência *Bugzilla*:** campo tipo *String*; campo não obrigatório;
- **Esforço alteração (h):** campo tipo *String*; campo não vazio;
    - O esforço deve ser definido em horas, ou seja, na unidade hora;
- **OK do Autor:** campo tipo *String*; campo não vazio;
    - Neste caso deve haver um ok ou um "x" para representar que o autor do registro de erro (inspeção) concluiu a sua inspeção;
- **OK do Coordenador:** campo tipo *String*; campo não vazio;
    - Neste caso também deve ser definido um ok ou um "x" para representar que o coordenador está informado da inspeção realizada;

Caso uma dessas regras não esteja sendo obedecida, é informada a inconsistência de determinado conceito, e, conseqüentemente, do documento corrente.

6.3. Conclusão

As regras de consistência são maneiras de expressar as restrições associadas às informações e de viabilizar a verificação da coerência das informações e auxiliam no tratamento estruturado das informações definidas em cada documento.

A descrição dos modelos de documentos, seus conceitos, atributos de conceitos e as regras de consistência são parte da contribuição do trabalho desenvolvido. Essas descrições

são necessárias para entender o processo de implementação da Estrutura de Documentos Textuais, e para compreender a forma de trabalho das ferramentas de análise que serão descritas no capítulo 7, pois cada documento possui suas próprias regras de consistência.

Os modelos detalhados neste capítulo, assim como conceitos e atributos de conceitos serão citados no próximo, pois a estrutura de documentos textuais criada agrega os modelos descritos.

O capítulo seguinte descreve a criação da estrutura de documentos textuais e as ferramentas de verificação de consistência no ambiente SEA.

7 Implementação dos Documentos Textuais Estruturados no Ambiente SEA

O presente capítulo apresenta as novas funcionalidades contempladas no ambiente SEA. Nesse contexto, documentos textuais estruturados foram inseridos ao ambiente com a finalidade de fornecer suporte ao projeto de *workflow* e melhorar a qualidade do processo de desenvolvimento de *software*.

As novas propriedades acrescentadas ao ambiente são facilitadas devido à flexibilidade do *framework* OCEAN, pois ele possui uma estrutura extensível, passível à criação de novas estruturas de especificação. Assim sendo, a nova estrutura pode ser adaptada a qualquer outro ambiente de desenvolvimento de *software* criado a partir do OCEAN. As funcionalidades existentes no ambiente SEA não se anulam em decorrência das novas adicionadas.

A estrutura de documentos textuais e o tratamento de requisitos de CMMI reforçam a possibilidade de melhora no processo de desenvolvimento de *software*, inseridos ao ambiente SEA, facilitando a organização das atividades e o registro das informações a serem estabelecidas no Projeto de *Workflow*, pois, uma atividade de inspeção, especificação de requisitos, plano de testes, formulário de aplicação de testes não era oferecida pelo ambiente SEA. Ressalta-se que outros documentos contemplados num processo poderiam fazer parte da estrutura de documentos textuais criada, mas essa extensão pode ser atendida em estudo posterior.

O diferencial de documentos textuais criados manualmente e a estrutura de documentos textuais fornecida, agora, pelo ambiente, é a estruturação desses documentos e a verificação da consistência dos documentos criados, por meio de ferramentas automatizadas que obedecem a regras de consistência para proceder a tal análise.

A seguir, são apresentados os passos da inserção da Estrutura de Documentos Textuais ao ambiente SEA e as ferramentas de análise de consistência dos documentos criados.

7.1 Estrutura dos Documentos inserida ao Ambiente SEA

O ambiente SEA não oferecia a capacidade de controlar o fluxo das atividades a serem realizadas durante o processo de desenvolvimento de *software*, suportando apenas o aspecto tecnológico do processo, fornecendo somente a produção de novas especificações, análise de consistência e geração de código.

Com a inclusão da abordagem de *workflow* (SCHEIDT, 2002) corrigiu-se a ausência de uma modelagem e gerenciamento de processos que o ambiente não fornecia, suprindo uma carência encontrada no mesmo. Foram, então, acrescentadas funcionalidades, seguindo a técnica de *workflow* e requisitos de CMMI permitindo, também, a otimização do processo e tratando parte do aspecto gerencial previsto no mesmo.

Apesar de ter sido corrigida uma deficiência encontrada no ambiente SEA com a contribuição de (SCHEIDT, 2002), a falta de registro de informações durante um processo ainda está presente no ambiente, pois, à medida que um Projeto de *Workflow* é criado, os processos (atividades) são definidos. Ao estabelecer um processo para criar um *Use Case* ou Diagrama de Classes, por exemplo, o SEA possui uma estrutura de especificação que oferece essas funcionalidades para tal execução.

Quando era estabelecida no Projeto de *Workflow* uma atividade de Inspeção ou uma criação de especificação de requisitos, por exemplo, o ambiente não fornecia subsídios para registrar as informações dentro do próprio ambiente. Nesse caso, apenas era informado que a inspeção havia sido realizada ou não, por exemplo, mas não havia uma forma semi-automatizada que pudesse facilitar essa verificação.

Para tratar essa carência do registro de informações e organização de processos a serem executados, foi criada uma estrutura de documentos textuais juntamente com requisitos de CMMI que pudessem ser adaptados a essa proposta, melhorando a qualidade do desenvolvimento de *software*.

A estrutura de documentos textuais criada contém quatro *templates* de documentos textuais estruturados: documento de especificação de requisitos, documento de plano de

testes, formulário de aplicação de testes, formulário de inspeção. Em conjunto com essa estrutura de documentos, foram criadas ferramentas de análise de consistência para verificar a coerência dos documentos criados a partir dessa estrutura. Ressalta-se que estes quatro documentos são apenas um ensaio de documentos textuais estruturados no ambiente SEA, mas nada impede de que outros documentos também façam parte dessa estrutura de especificação.

O procedimento da inclusão da nova funcionalidade ao ambiente SEA ocorre com a extensão do *framework* OCEAN que suporta diferentes tipos de estruturas, como já mencionado.

A Estrutura de Documentos Textuais criada insere o princípio de documentar os processos durante o desenvolvimento, prática utilizada principalmente em empresas que querem melhorar seus processos e/ou adotar algum modelo de qualidade do processo de desenvolvimento. O princípio citado acima cria subsídios para que possam documentar atividades de requisitos de CMMI que demandam documentação.

Para contemplar esses documentos com a verificação de consistência das informações dos mesmos, foram criadas ferramentas de consistência, que possuem a finalidade de analisar a coerência das informações dos documentos criados e da especificação que está sendo tratada.

7.2 Criação da Especificação: Estrutura de Documentos Textuais no Ambiente SEA

Como comentado, diferentes estruturas de especificação podem ser criadas devido à flexibilidade do *framework* OCEAN, permitindo que a inclusão de nova especificação ao ambiente SEA seja possível.

Para implementar a Estrutura de Documentos Textuais no ambiente, foi criada uma nova estrutura de especificação a partir do *framework* OCEAN. Assim, quando uma especificação é criada é apresentada na janela do ambiente a relação de todas os tipos de especificações oferecidos pelo ambiente de desenvolvimento de *software*. Como foi

adicionada uma nova especificação Estrutura de Documentos Textuais, essa fará parte da relação, como demonstra a figura 7.1.

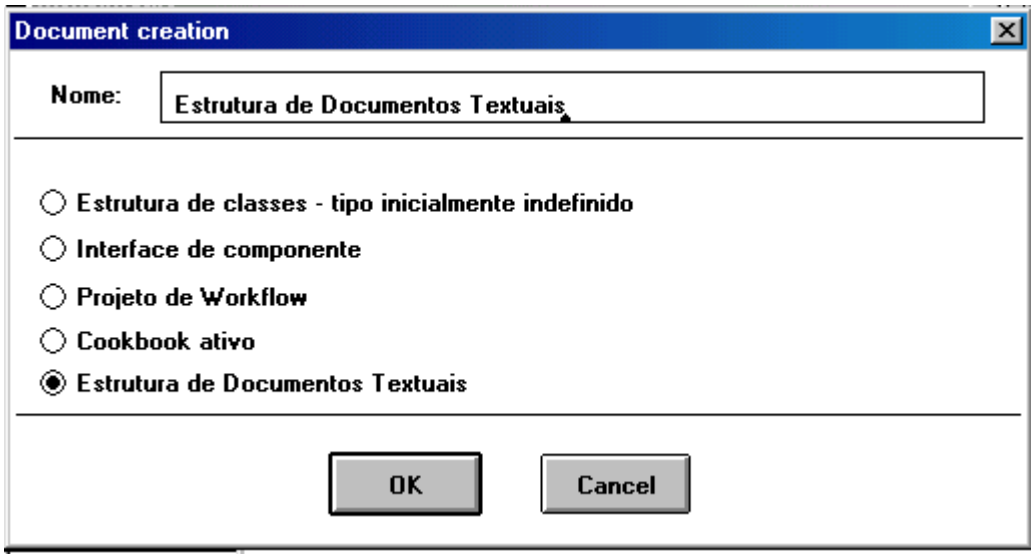


Figura 7.1 - Janela de especificações com a nova especificação do ambiente SEA

A figura 7.1 exemplifica um exemplo da seleção de uma especificação e a definição do nome para ela. No caso, a especificação selecionada é Estrutura de Documentos Textuais. A relação dos modelos suportados por essa especificação, é descrita na próxima seção.

7.3 Estrutura de Documentos Textuais – Modelos e Conceitos

Uma especificação agrega elementos de especificação denominados modelos e conceitos e permite associações de sustentação e referência entre dois elementos de especificação.

Para construir uma estrutura de especificação, um conjunto de tipos de modelos e conceitos deve ser definido agregando-os a essa estrutura e associando seus elementos. Cada modelo é uma subclasse concreta de *ConceptualModel* e cada conceito é uma subclasse concreta de *Concept*, classes definidas no *framework*.

Foi definido, então, que cada documento Textual estruturado (descrito no capítulo 6) seria tratado como modelo. Portanto, cada um dos modelos,

DocumentoEspecificacaoRequisitos, DocumentoPlanoTeste, FormularioAplicacaoTeste e FormularioInspecao, é subclasse concreta de *ConceptualModel*.

Assim, cada elemento associado aos modelos (descrito no capítulo 6) foi definido como conceito. Por isso, cada conceito, IdentificadorDoc, HistoricoDocumento, ItemHistorico, Introducao, VisaoGeralSistema, ConjuntoRequisitos, Requisito, CasosTeste, CasoTeste, Identificação, Agendamento, Colaboradores, PessoaEnvolvida, ResultadoAplicacaoTeste, RegistroTeste, Inspeção e Erro é subclasse concreta de *Concept*. Os modelos e conceitos expostos são acoplados ao *framework* OCEAN. A figura 7.2 demonstra como as subclasses dos modelos e dos conceitos da nova especificação são inseridos ao OCEAN.

Ressalta-se que as figuras 7.2, 7.3, 7.8, 7.22 são representações das classes e subclasses do *framework* OCEAN e que as cores utilizadas são para distinguir classes já existentes no OCEAN e as subclasses criadas a partir do mesmo como contribuição deste trabalho. Contudo, as classes de cor amarela representam as classes do OCEAN e as subclasses de cor pêssego são as criadas para a Estrutura de Documentos Textuais. Os nomes das classes utilizados são os mesmos definidos no *framework*, tanto os das classes existentes quanto das que foram criadas neste estudo.

Dentro do contexto apresentado e, partindo do princípio de associações de sustentação descrito no capítulo 3, dentro da nova estrutura de especificação definida, existe uma associação de sustentação entre os documentos plano de teste e o formulário de aplicação de teste no qual o elemento sustentador é o documento de plano de teste e o sustentado é o formulário de aplicação de testes.

Lembrando que, um formulário de aplicação de teste só pode ser criado caso exista um plano de teste associado a ele, pois o elemento resultado de aplicação de teste (conceito referenciado pelo modelo formulário de aplicação de teste) é sustentado pelo elemento casos de teste (conceito referenciado pelo modelo documento de plano de teste). Assim, um resultado de aplicação de teste só existirá, se houver um CasosTeste associado a ele.

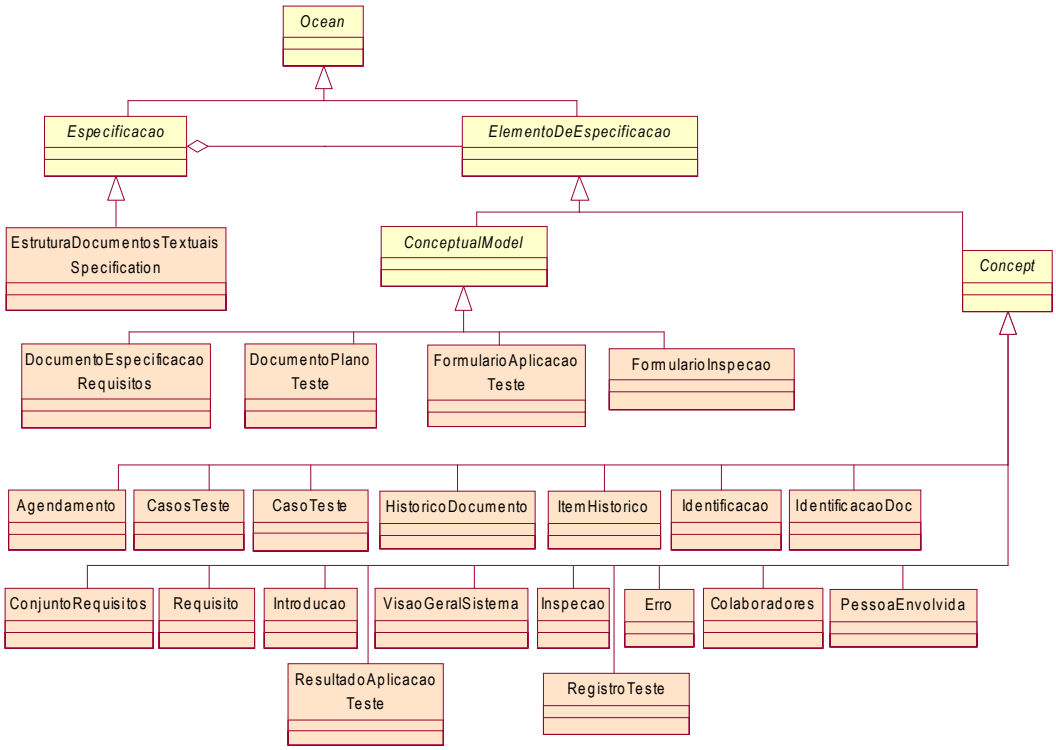


Figura 7.2 - Estrutura das subclasses dos conceitos e modelos

7.4 Visualização de Documentos no Ambiente SEA

Ao incluir uma nova especificação na estrutura do ambiente SEA, novos elementos de especificação (modelos e conceitos) são definidos, assim expostos no ambiente. Mas, para que esses elementos se tornem visíveis precisam ser manipulados graficamente.

Como descrito no capítulo 2, o OCEAN dispõe de uma classe gerenciadora *ReferenceManager* que associa cada elemento de especificação a um mecanismo de visualização. Para essa representação visual, O OCEAN criou subclasses das classes do *framework HotDraw* (BRANDT, 1995), pois este *framework* permite criar e editar elementos visuais. Por isso, todos os elementos de especificação criados estão associados ao *HotDraw*, e, com isso, para cada conceito é inserido uma figura do conceito.

A classe *ReferenceManager* do OCEAN referencia a subclasse *SpecificationElementInterface* que contém as subclasses *ConceptInterface* (estrutura genérica

dos mecanismos de visualização de conceitos*)* e *ConceptModelInterface* (estrutura genérica dos mecanismos de visualização de modelos).

Para os modelos pertencentes à nova especificação criada, Estrutura de Documentos Textuais, foi criado um editor para cada um dos modelos. Assim, foi inserida quatro classes referentes a cada modelo DocumentoEspecificacaoRequisitos, DocumentoPlanoTeste, FormularioPlanoTeste e FormularioInspecao, classes concretas de *ConceptualModel* (conforme apresentada na figura 7.2) e subclasses concretas de *SpecificationCompositeFigure*, IdentificadorDocFigure, AgendamentoFigure, CasosTesteFigure, HistoricoDocumentoFigure, IdentificacaoFigure, InspecaoFigure, IntroducaoFigure, ConjuntoRequisitosFigure, ColaboradoresFigure, ResultadoAplicacoTesteFigure, VisaoGeralSistemaFigure.

Existe a necessidade de uma associação entre a parte gráfica e a parte semântica dos elementos (subclasses de *SpecificationCompositeFigure*- descritas acima - figuras dos conceitos) do modelo, para que qualquer modificação realizada na parte gráfica interfira na estrutura semântica do mesmo. Devido a essa necessidade, as subclasses estão relacionadas às subclasses de *SpecificationElement*.

A estrutura das subclasses criadas a partir do *framework* para a criação dos editores de cada modelo é demonstrada na figura 7.3.

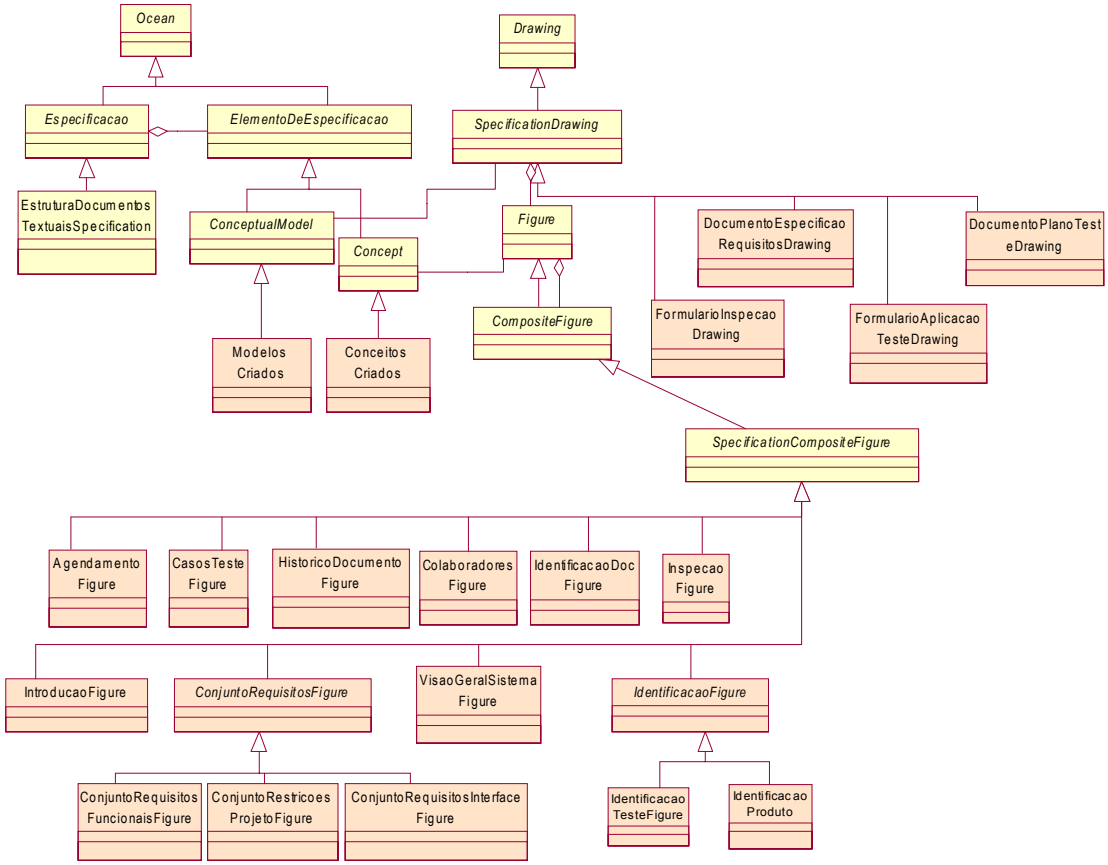


Figura 7.3 - Estrutura das subclasses dos editores gráficos a partir do OCEAN

A figura 7.3 apresentou a estrutura das subclasses de um editor gráfico a partir do *framework* OCEAN.

Como apresentado na figura 7.3, para cada modelo criado numa especificação, é necessário criar subclasses de *SpecificationFigure*, que por sua vez, é subclasse de *DrawingEditor*. Os editores dos modelos da especificação Estrutura de Documentos Textuais são: DocumentoEspecificacaoRequisitosEditor, DocumentoPlanoTesteEditor, FormularioAplicacaoTesteEditor, FormularioInspecaoEditor e a estrutura dessas classes será demonstrada na figura 7.8, juntamente com a estrutura das classes das janelas de edição dos modelos e conceitos.

As figuras 7.4, 7.5, 7.6, 7.7 apresentam os editores dos respectivos modelos: DocumentoEspecificacaoRequisitos, DocumentoPlanoTeste, FormularioPlanoTeste e

FormularioInspecao. Cada modelo (editor de modelo) possui seus próprios conceitos. Ressalta-se que um mesmo conceito pode ser referenciado por modelos diferentes.

Para cada modelo, os conceitos referenciados por ele são representados através de letras que possam identificar o conceito tratado. Os conceitos pertencentes ao documento de especificação de requisitos possuem as seguintes representações, conforme se observa na figura 7.4.

- **ID**: Identificador do documento;
- **H**: Histórico do Documento;
- **I**: Introdução;
- **VG**: Visão geral do sistema;
- **RF**: Conjunto de requisitos funcionais;
- **RI**: Conjunto de requisitos de interface;
- **RP**: Conjunto de restrições de projeto;

Ressalta-se que, cada "identificador" do conceito foi determinado durante o desenvolvimento do trabalho e que, poderia ser representado por outras letras que pudessem identificar tal conceito.

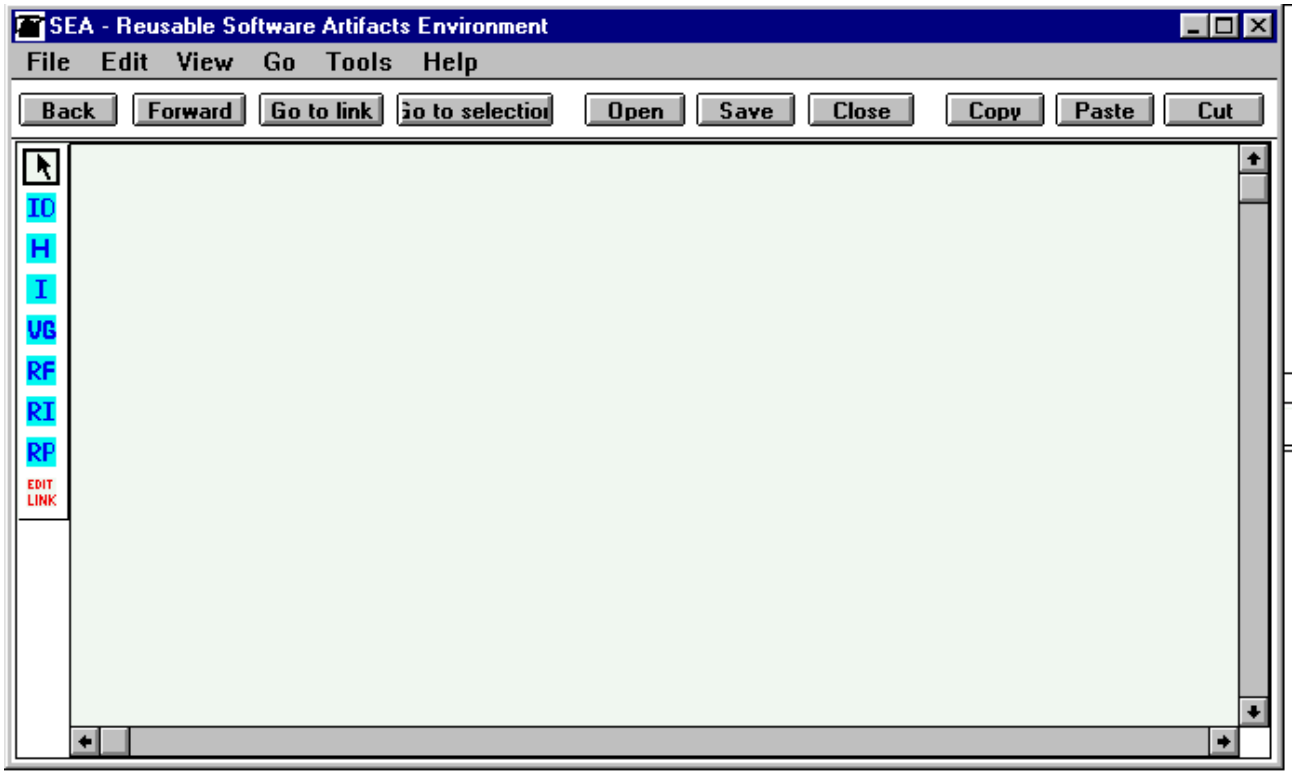


Figura 7.4 – Editor do Modelo Documento de Especificação de Requisitos

No modelo documento de plano de teste, alguns conceitos também são referenciados pelo documento de especificação de requisitos como mencionado no capítulo 6. Os conceitos referenciados pelo documento de plano de teste são identificados pelas seguintes letras e seus respectivos significados, conforme observa-se na figura 7.5.

- **ID**: Identificador do documento;
- **H**: Histórico do Documento;
- **I**: Introdução;
- **CT**: Casos de Teste;

Assim como no documento de especificação de requisitos, o documento de plano de teste também poderia ter outros "identificadores" dos conceitos ao invés dos que foram definidos.

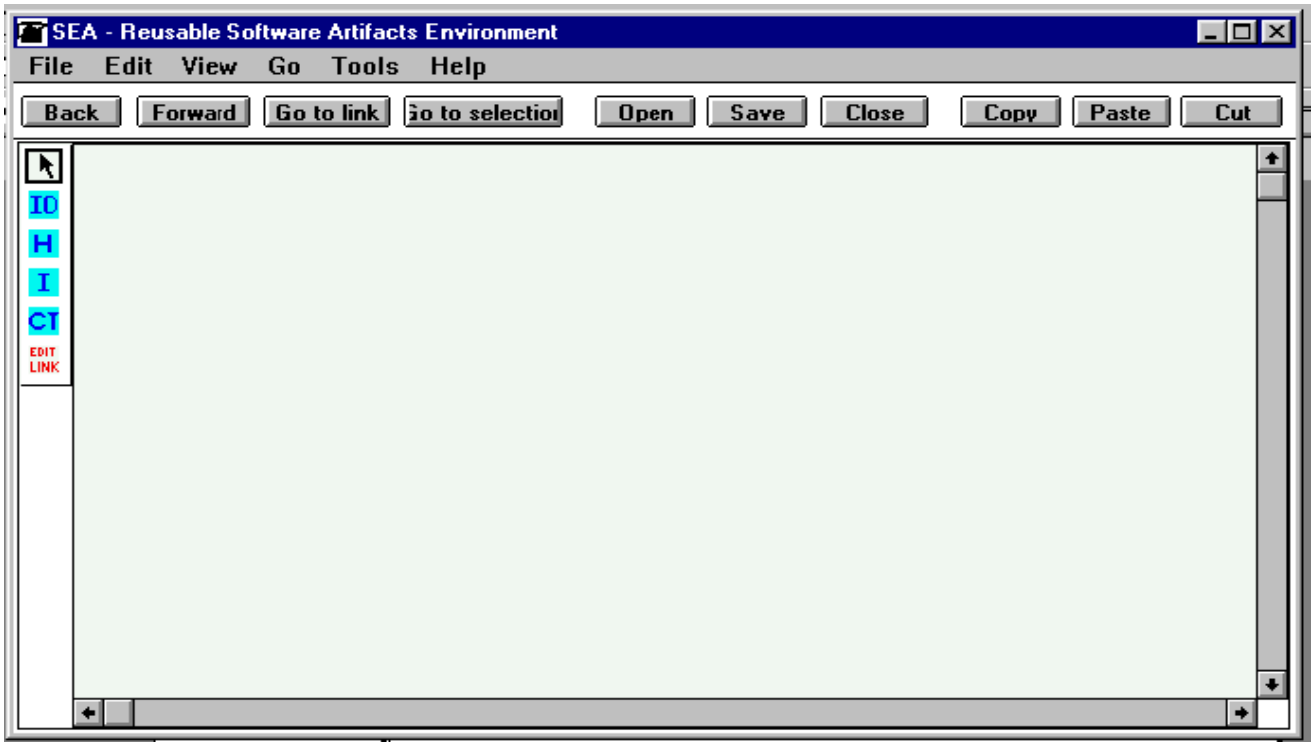


Figura 7.5 - Editor do Modelo Documento de Plano de Teste

Os conceitos referenciados pelo documento formulário de aplicação de teste são identificados pelas seguintes letras e símbolos com seus respectivos significados, conforme apresenta a figura 7.6.

- **IT**: Identificação (Teste/Produto);
- : Agendamento;
- **PE**: Colaboradores;
- **RT**: Resultado de aplicação do teste;

Da mesma forma que o documento de especificação de requisitos e de plano de teste, o formulário de aplicação de teste possui "identificadores" dos conceitos que ao invés de IT para identificação, RT para resultado de aplicação, poderiam ser outros.

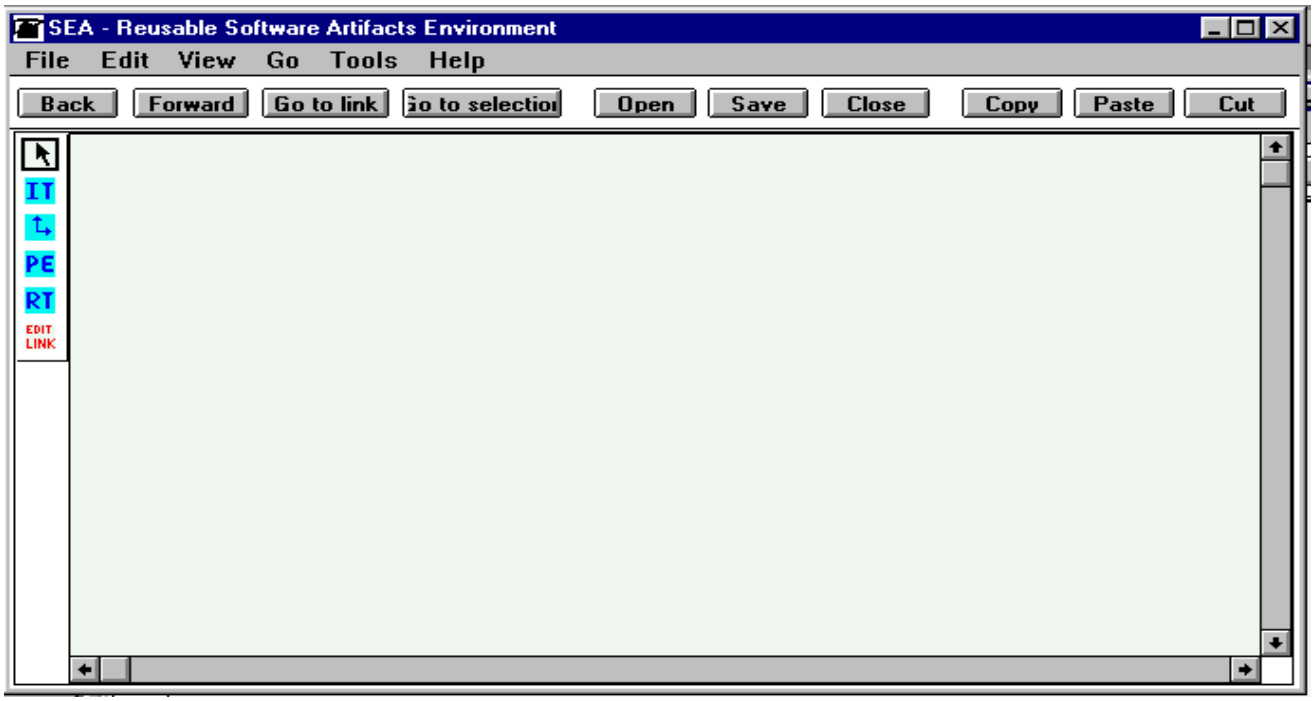


Figura 7.6 – Editor do Modelo Formulário de Aplicação de Teste

No modelo formulário de inspeção, alguns conceitos também são referenciados pelo documento formulário de aplicação de teste como também mencionado no capítulo 6. Os conceitos referenciados pelo formulário de inspeção são identificados pelas seguintes letras e símbolos com seus significados, conforme se observa na figura 7.7.

- **IT**: Identificação (Teste/Produto);
-  : Agendamento;
- **PE**: Colaboradores;
- **RS**: Inspeção;

Assim como os modelos demonstrados anteriormente, o formulário de inspeção poderia ter tido outros identificadores para os conceitos ao invés dos apresentados acima.

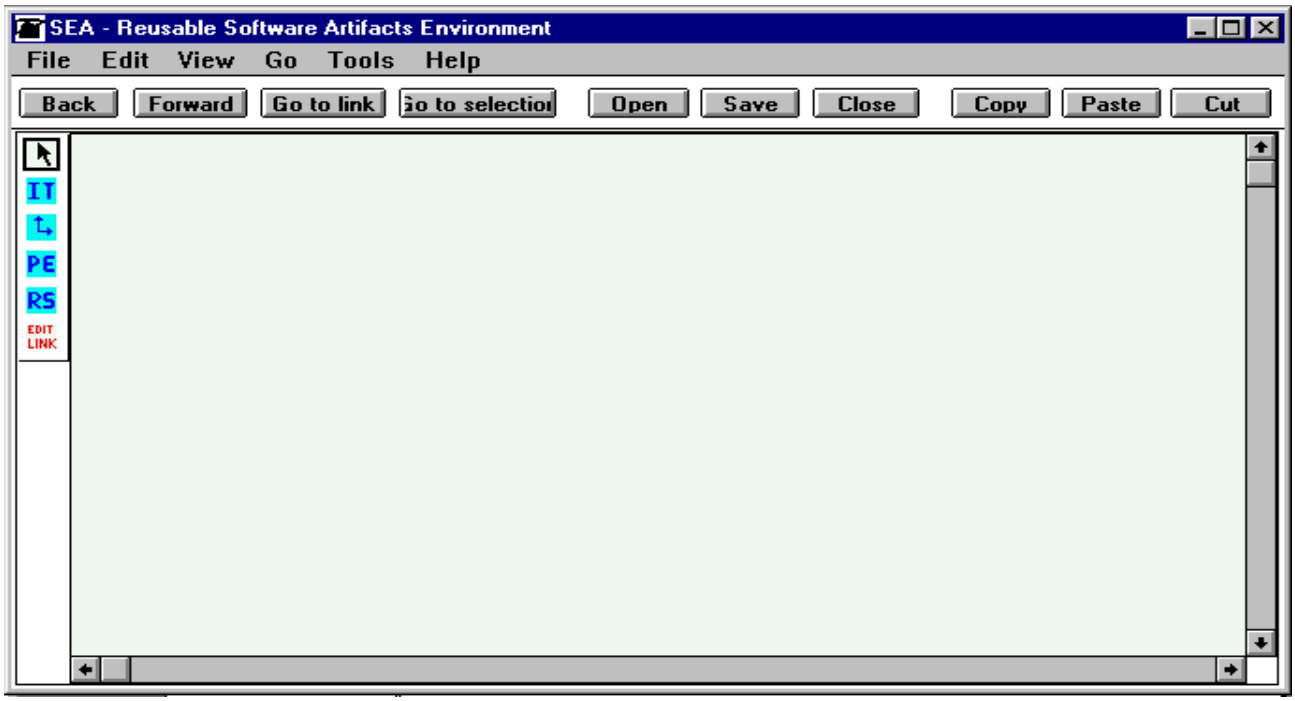


Figura 7.7 - Editor do Modelo Formulário de Inspeção

As figuras 7.4, 7.5, 7.6 e 7.7, demonstraram os editores dos modelos de documentos textuais disponíveis na especificação Estrutura de Documentos Textuais.

Para cada conceito, ao lado esquerdo de cada editor, existe uma janela de edição responsável por fornecer as informações a serem procedidas em cada um dos conceitos. Essas janelas são descritas no próximo item.

7.5 Janelas de Edição dos Conceitos da Estrutura de Documentos Textuais

Para cada conceito na Estrutura de Documentos Textuais existe uma janela de edição responsável por fornecer as informações a serem definidas para cada conceito.

As classes referentes às janelas dos conceitos são subclasses de *ConceptAuxWindow*, subclasse de *ApplicationModel*.

As janelas de edição dos conceitos possuem uma associação entre as classes dos editores dos modelos (subclasse de *SpecificationEditor*, que é subclasse *DrawingEditor*) e as

classes das janelas de cada modelo (subclasse de *EditorAuxWindow*), pois estas são subclasses de *Model* na hierarquia mais alta. As classes *EditorAuxWindow* (para criação de subclasses das janelas dos modelos) e *ConceptAuxWindow* (para criação de subclasses das janelas dos conceitos) são subclasses de *ApplicationModel* que, por sua vez é subclasse de *Model*.

A figura 7.8 apresenta a estrutura da criação das janelas de edição dos modelos e janelas de edição dos conceitos.

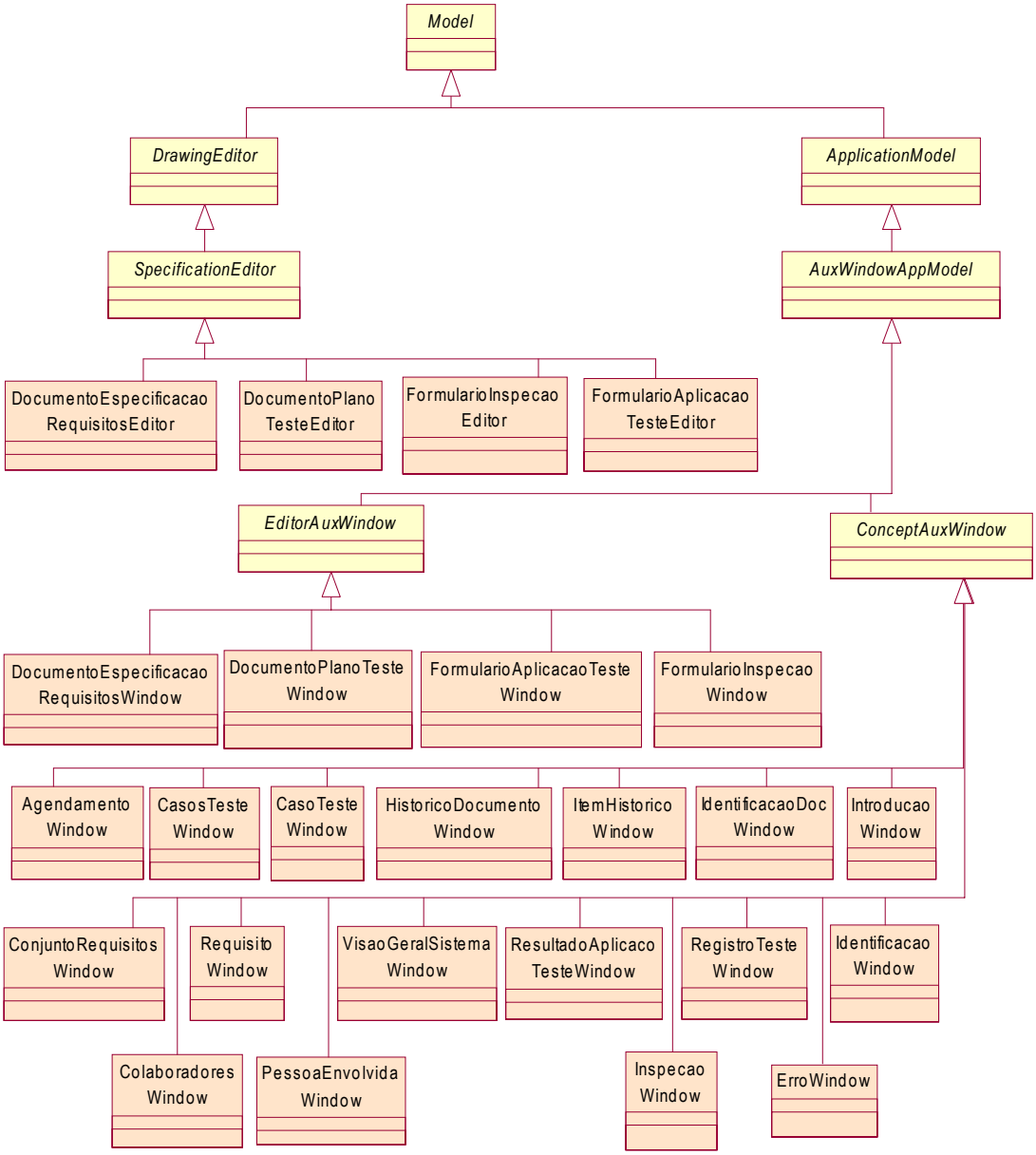


Figura 7.8 - Estrutura dos editores de modelos associados às janelas de edição dos mesmos conjuntamente com as janelas de edição dos conceitos

As figuras seguintes apresentam as janelas de edição de cada conceito do documento de especificação de requisitos presente na estrutura de documentos textuais.

A figura 7.9 demonstra a janela de edição do conceito identificador do documento, escolhendo a opção "*Atributes*" da janela. A opção *Identify* é a propriedade que demonstra o tipo de conceito e na escolha da opção "*Links*", pode-se criar um *link* para uma especificação, um documento. A opção "*Links*" possui a mesma finalidade para os outros conceitos da Estrutura de Documentos Textuais.

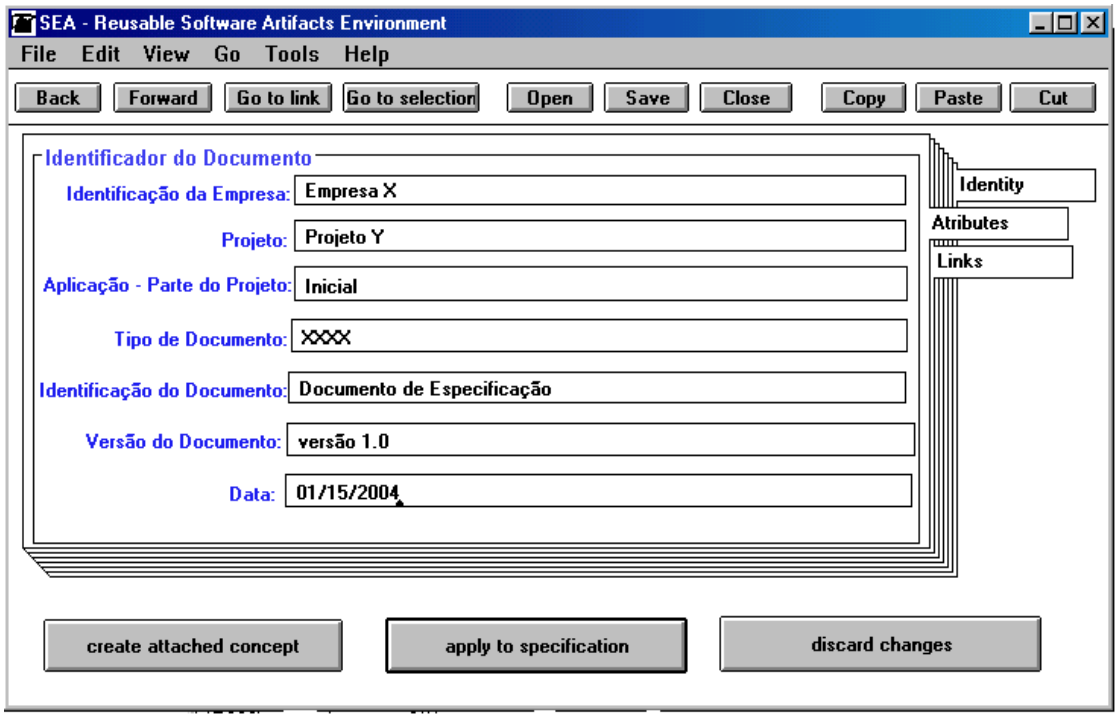


Figura 7.9 - Janela de edição do conceito identificador do documento

A figura 7.10 apresenta a janela de edição do conceito histórico do documento. Nessa janela é apresentada uma coleção de itens de históricos representada pela chave que, nesse caso, é a versão. Essa coleção é obtida por meio de um conceito referenciado pelo histórico do documento. O conceito referenciado permite registrar os itens de históricos do documento corrente. As informações a serem definidas nesse conceito foram apresentadas no capítulo 6.

Ao escolher a opção Incluir Histórico da janela de edição do conceito histórico do documento conforme observa-se na figura 7.10, uma outra janela de edição é referenciada. A figura 7.11 apresenta as informações a serem procedidas.

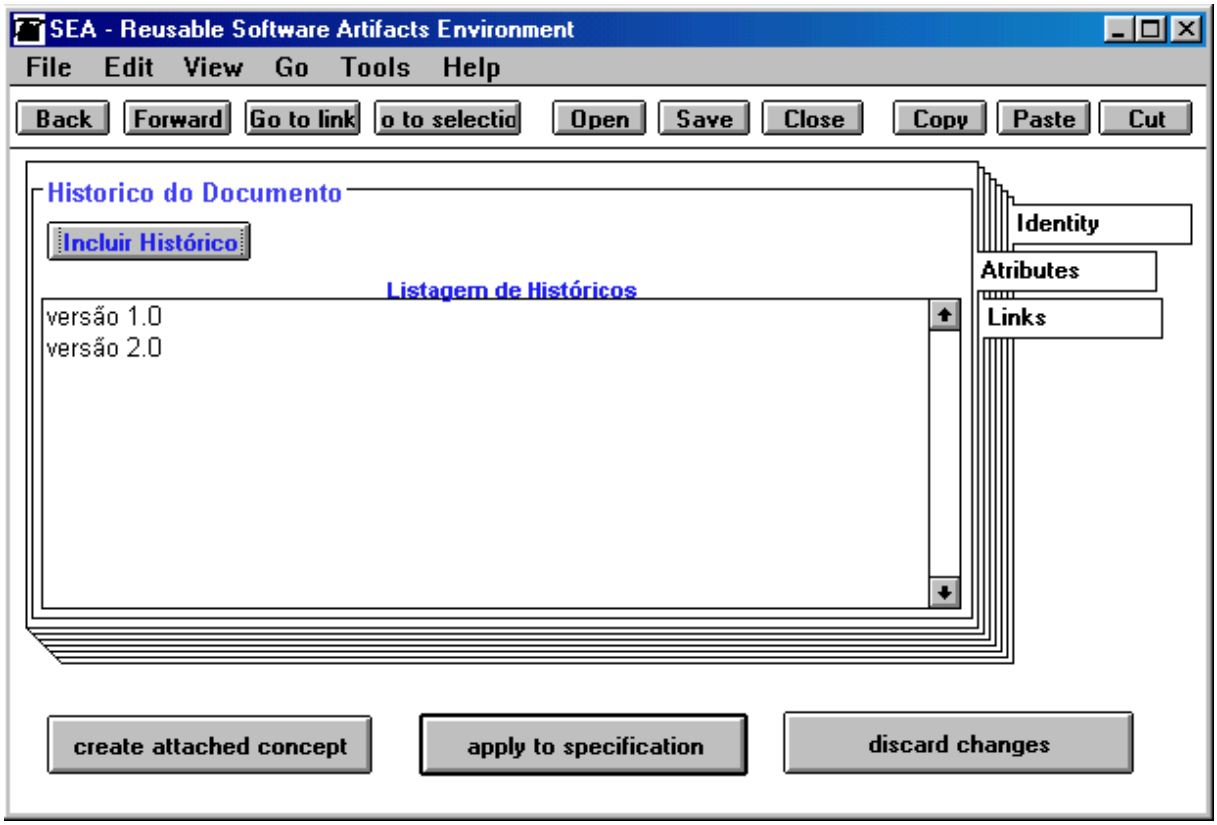


Figura 7.10 - Janela de edição do conceito histórico do documento

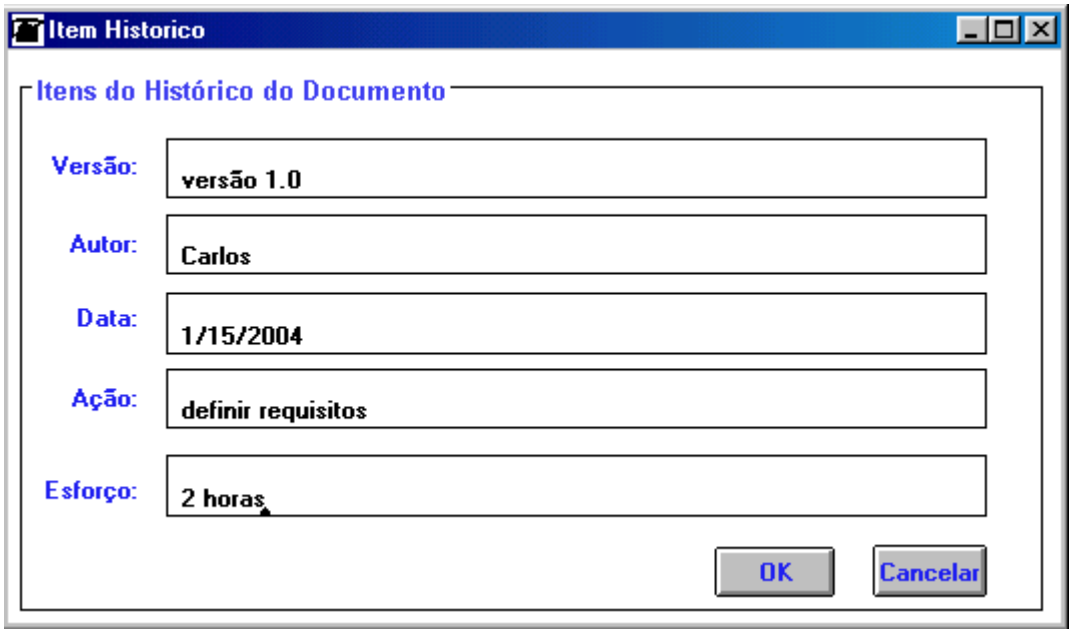


Figura 7.11 - Janela de edição da opção Incluir Histórico da janela do conceito histórico do documento

A figura 7.11 apresenta os atributos, ou seja, informações que devem fazer parte de um histórico de documento.

A figura 7.12 apresenta a janela de edição do conceito visão geral do sistema e as respectivas informações a serem definidas por meio da sua janela de edição.

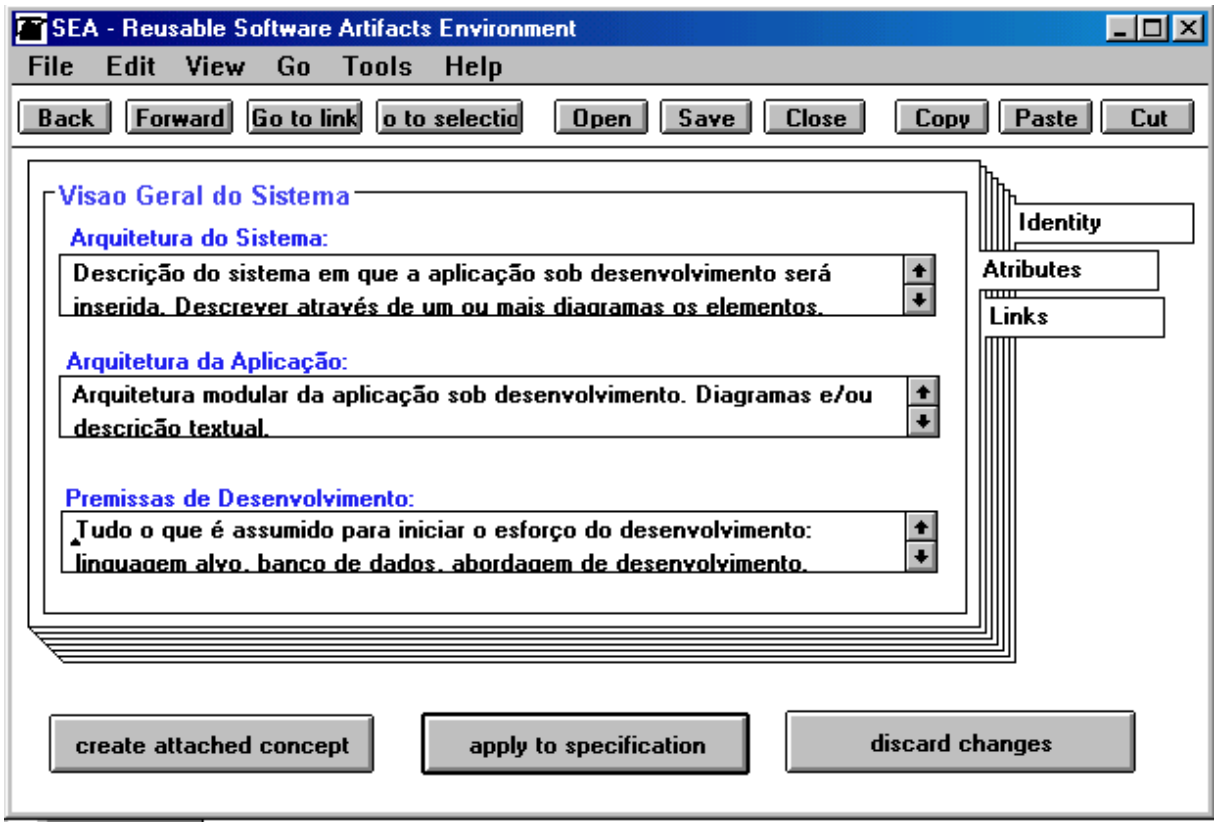


Figura 7.12 - Janela de edição do conceito visão geral do sistema

A figura 7.13 apresenta a janela de edição do conceito introdução e as respectivas informações a serem definidas por meio da janela de edição. Algumas informações desse conceito são essenciais e, por isso, existem regras de consistência para tais informações. As regras das informações essenciais desse conceito estão descritas no capítulo 6.

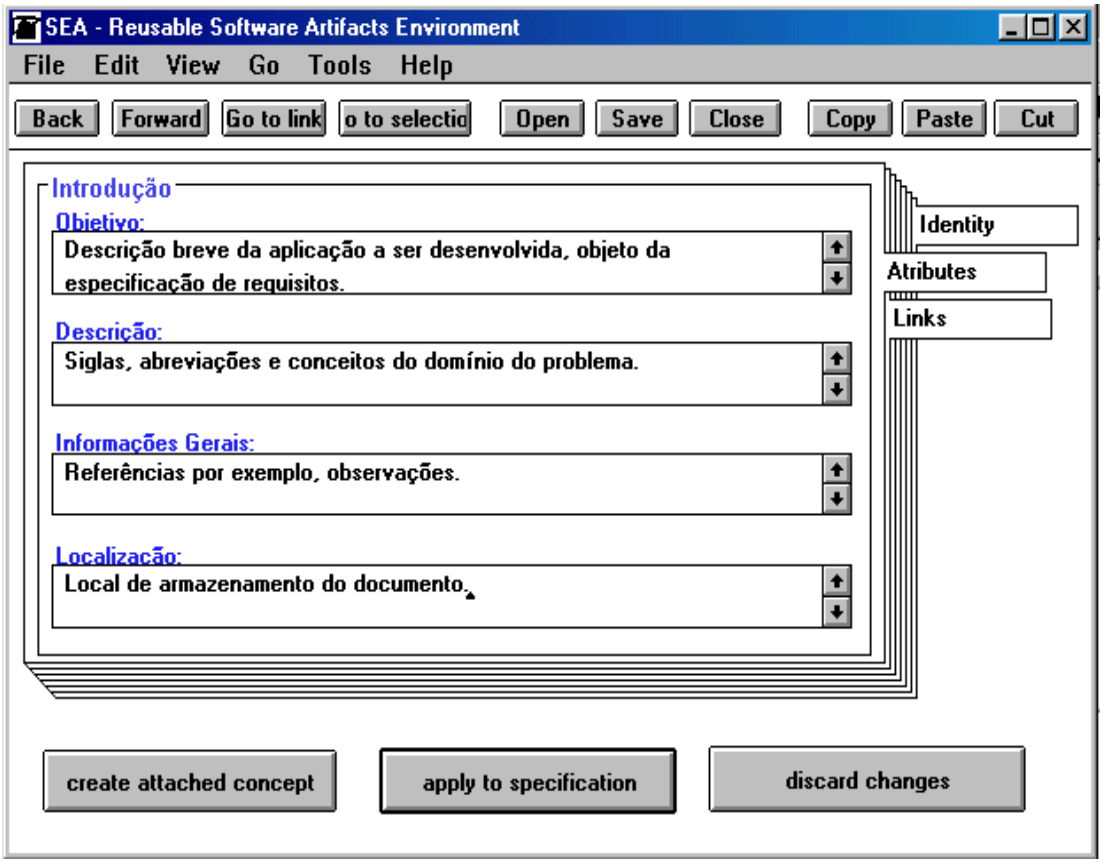


Figura 7.13 - Janela de edição do conceito introdução

A figura 7.14 apresenta a janela de edição do conceito conjunto de requisitos. Esse conceito contém a lista dos requisitos que são definidos no documento de especificação de requisitos. Essa lista é obtida por meio da janela de edição do conceito referenciado pelo conjunto de requisitos. A figura 7.14 exemplifica o conceito requisitos funcional.

Ao escolher a opção Incluir Requisito da janela de edição do conceito conjunto de requisitos conforme apresenta a figura 7.14, uma outra janela é referenciada, a janela do conceito Requisito, contendo informações necessárias a serem procedidas. A figura 7.15 apresenta a janela referenciada, ou seja, a janela de edição dos requisitos, no caso dessa figura, o exemplo é de criação de um requisito funcional. Esse conceito possui informações essenciais e, por isso, existem regras de consistência para essas informações. Essas regras foram detalhadas no capítulo 6.

Tanto a janela do conceito ConjuntoRequisitos e Requisito são iguais para as três instâncias do conceito ConjuntoRequisitos.

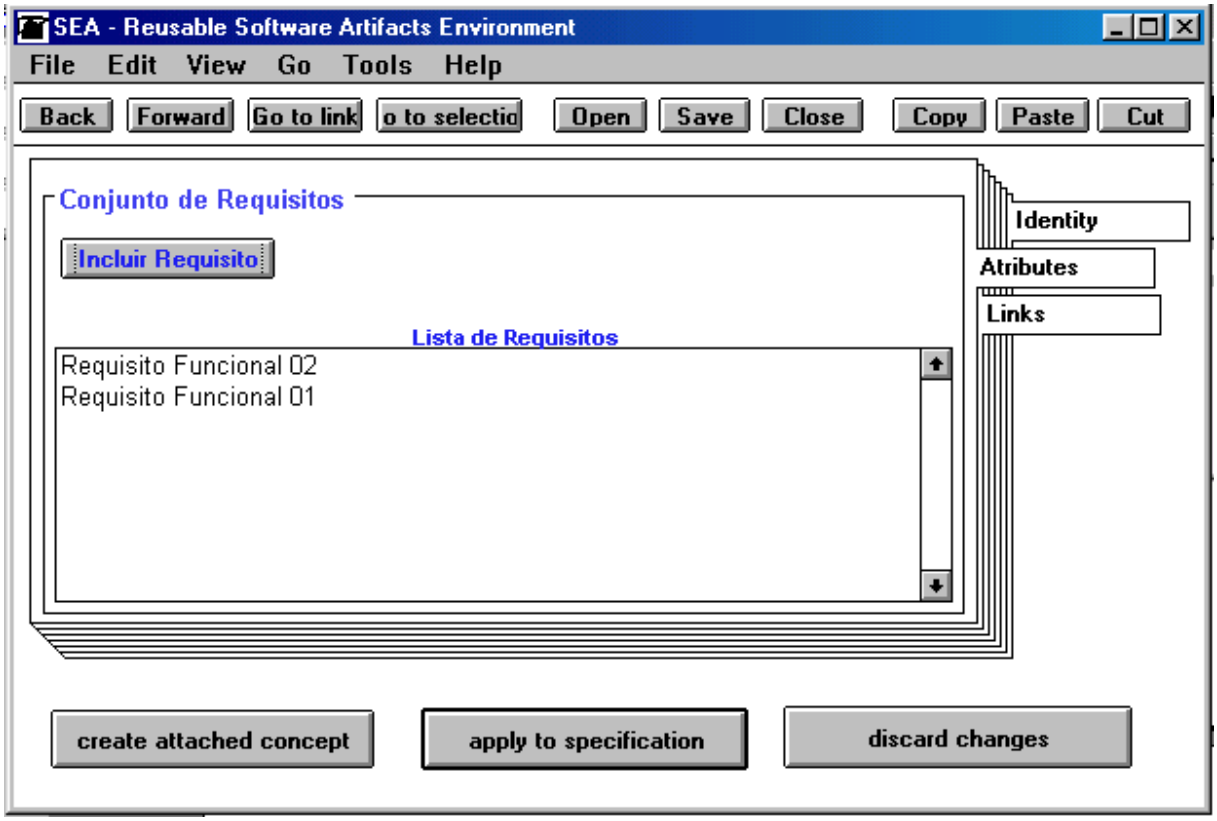


Figura 7.14 - Janela de edição do conceito conjunto de requisitos – exemplo funcionais

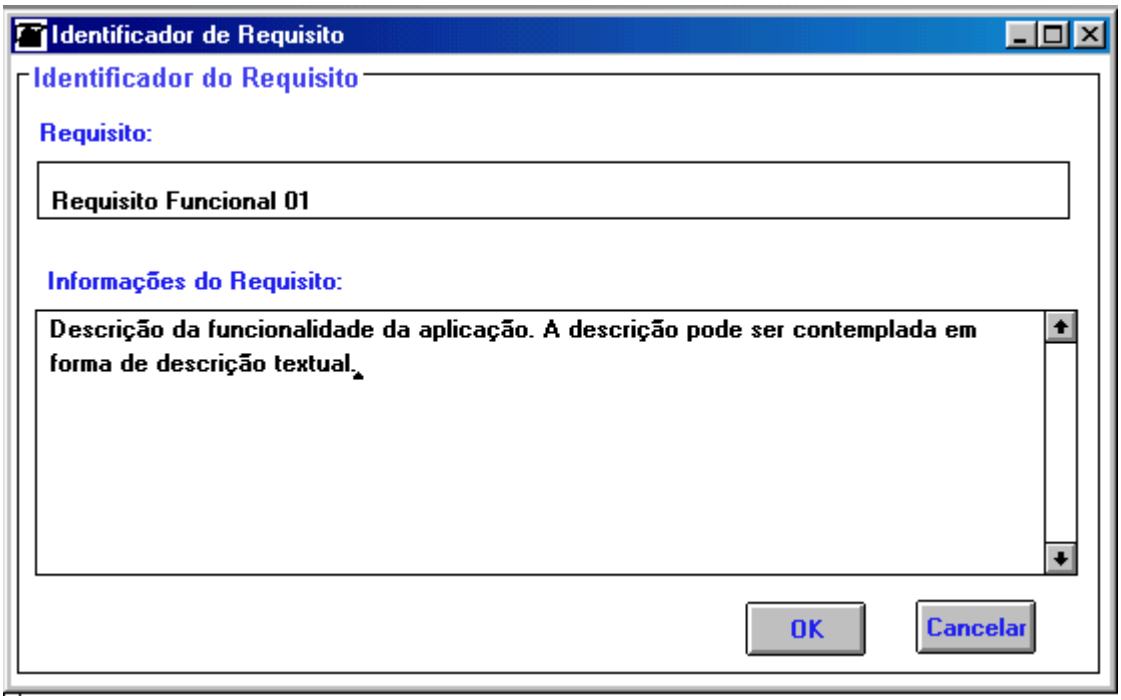


Figura 7.15 - Janela de edição do conceito requisito – exemplo requisito funcional

A figura 7.16 apresenta um exemplo da coleção de requisitos de interface através da janela de edição ConjuntoRequisitos obtido por meio da janela do conceito referenciado pelo

mesmo. No caso desse exemplo foi criado apenas um requisito de interface. Nessa janela é demonstrada a coleção dos requisitos de interface definidos.

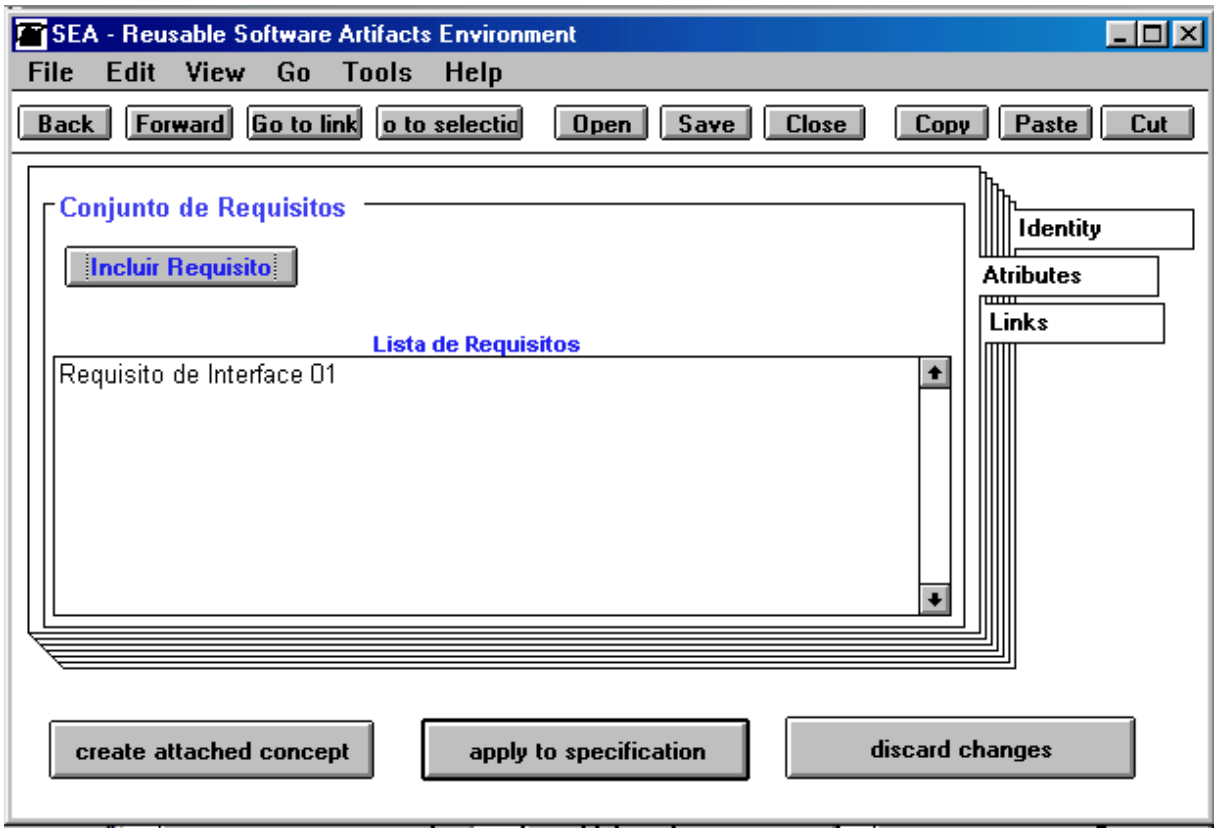


Figura 7.16 - Janela de edição do conceito de requisitos – exemplo de interface

Ao escolher a opção Incluir Requisito da janela de edição do conceito conjunto de requisitos, da figura 7.16, é possível a criação de um requisito de interface, conforme figura 7.17.

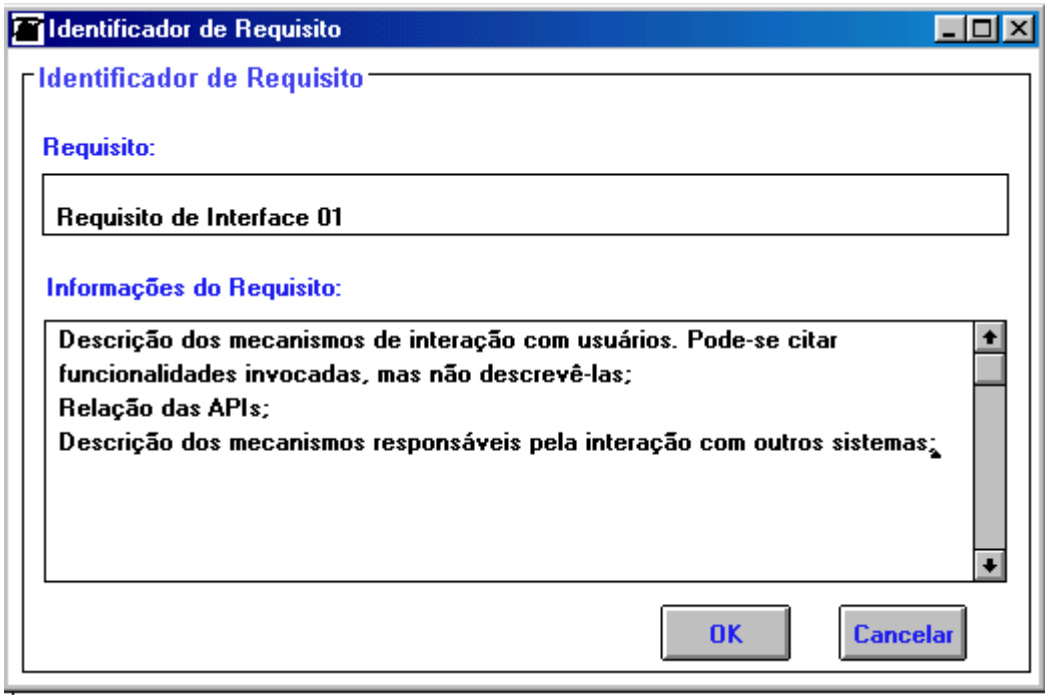


Figura 7.17 - Janela de edição do conceito requisito – exemplo requisito de interface

A figura 7.18 apresenta o exemplo da coleção de restrições de projeto. Esse exemplo demonstra a lista de restrições de projeto que forma. Essa lista é obtida por meio da mesma janela que os outros exemplos. A janela de edição do conceito referenciado, restrição de projeto, é apresentada na figura 7.19.

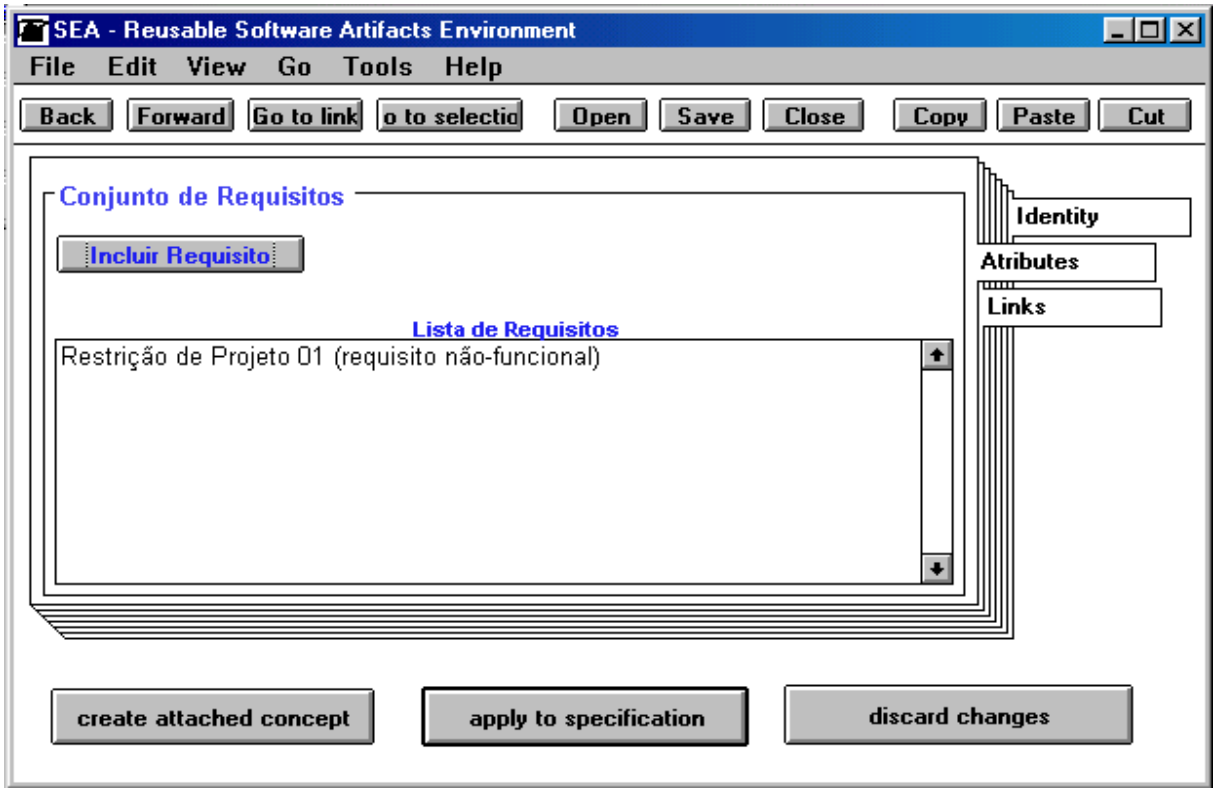


Figura 7.18 - Janela de edição do conceito conjunto de requisitos – exemplo de restrições de projeto

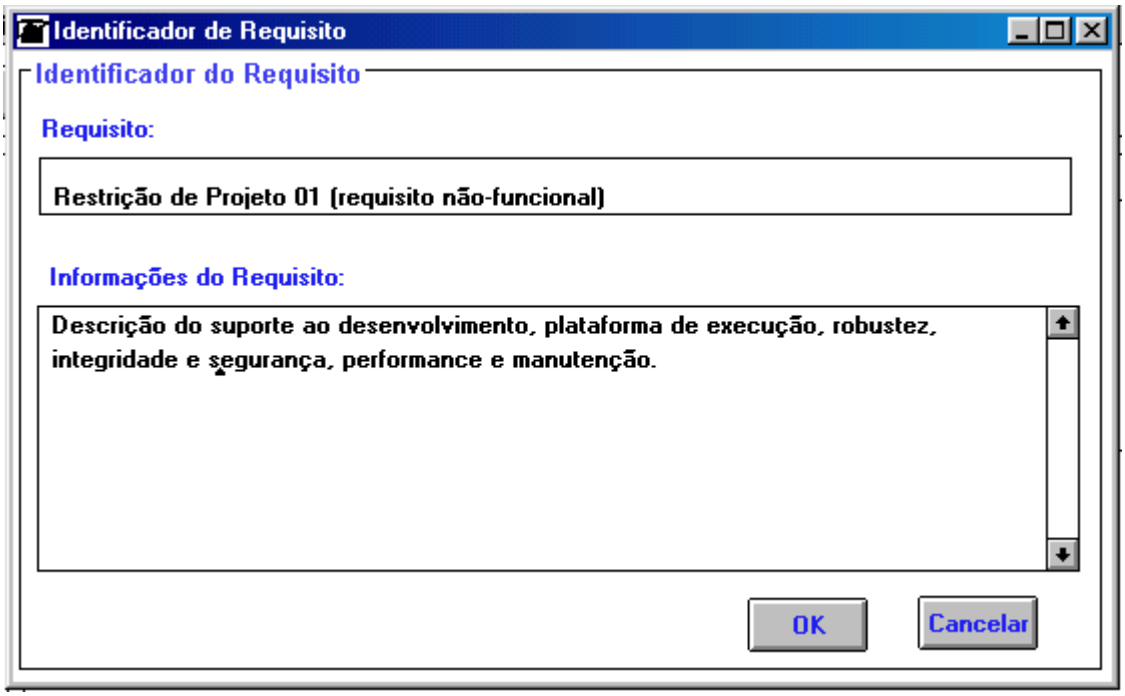


Figura 7.19 - Janela de edição do conceito – exemplo restrição de projeto

As figuras 7.9, 7.10, 7.11, 7.12, 7.13, 7.14, 7.15, 7.16, 7.17, 7.15, 7.16, 7.17, 7.18 e 7.19 apresentaram as janelas de edição dos conceitos que fazem parte do documento de especificação de requisitos.

A figura 7.20 apresenta a janela de edição das informações do documento de especificação de requisitos. Para cada conceito existe um "retângulo" contendo informações sobre o mesmo. Nem todos os retângulos dos conceitos estão aparecendo, mas para verifica-los basta interagir com a barra de rolagem.

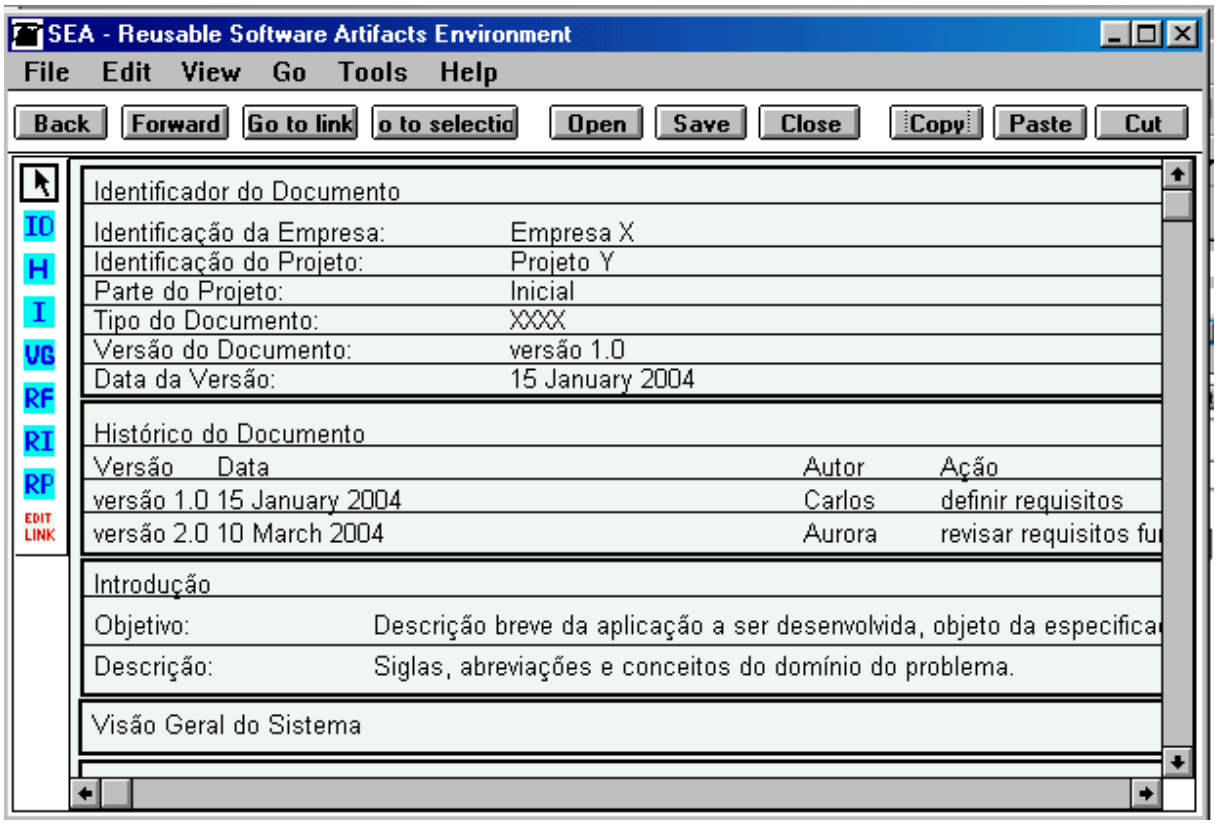


Figura 7.20 – Janela de edição do documento de especificação de requisitos.

A janela de edição da opção "*Links*" é a mesma para todas. Escolhendo essa opção é possível criar um *link* entre o conceito do documento corrente e uma especificação/outro documento. A figura 7.21 apresenta a janela de edição da opção "*Links*".

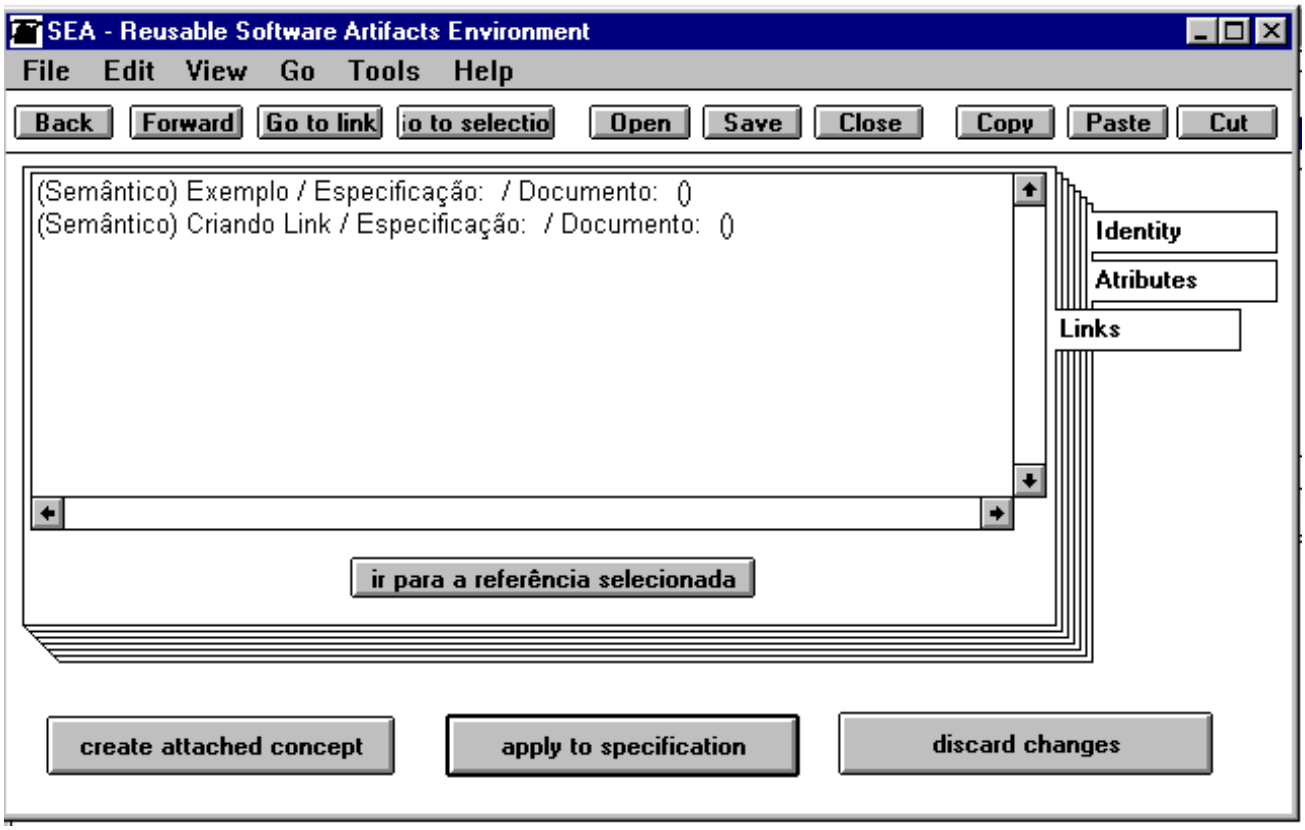


Figura 7.21 - Janela de edição da opção *Links*

As informações a serem preenchidas por meio dos campos da opção "*Atributes*" da janela de edição de cada conceito são contempladas com uma verificação de consistência nos seus documentos (modelos e especificação). Essa verificação ocorre através de ferramentas que analisam a consistência das informações, verificando primeiro a criação do modelo e, também, das informações estabelecidas em cada conceito. Essas ferramentas seguem um conjunto de regras que obedecidas em conjunto tornam um documento consistente.

As ferramentas que são responsáveis pela análise de consistência dos modelos são descritas a seguir.

7.6 Ferramentas de Análise de Consistência dos Modelos da Estrutura de Documentos Textuais

As ferramentas de análise de consistência são responsáveis por verificar a consistência das informações estabelecidas em cada documento, e reutilizam os procedimentos de edição semântica oferecidos pelo OCEAN.

Todas as ferramentas criadas, são subclasses concretas de *OceanTool* (classe do *framework* OCEAN) e podem estar associadas a um ou mais tipos de especificação e a um ou mais tipos de modelos.

No caso da estrutura de documentos textuais, foi criada uma ferramenta de análise de consistência para cada modelo construído nessa estrutura e uma ferramenta para analisar toda a especificação.

As ferramentas de análise de cada modelo denominam-se Ferramenta de Análise de Documento de Especificação de Requisitos (FerramentaAnaliseDocEspecRequisitos), Ferramenta de Análise do Documento de Plano de Teste (FerramentaAnaliseDocPlanoTeste), Ferramenta de Análise do Formulário de Aplicação de Teste (FerramentaAnaliseFormAplicacaoTeste), Ferramenta de Análise do Formulário de Inspeção (FerramentaAnaliseFormInspecao) e Ferramenta de Análise da Especificação (SEA3Analyser).

A figura 7.22 apresenta a estrutura da criação das ferramentas sob o *framework* OCEAN.

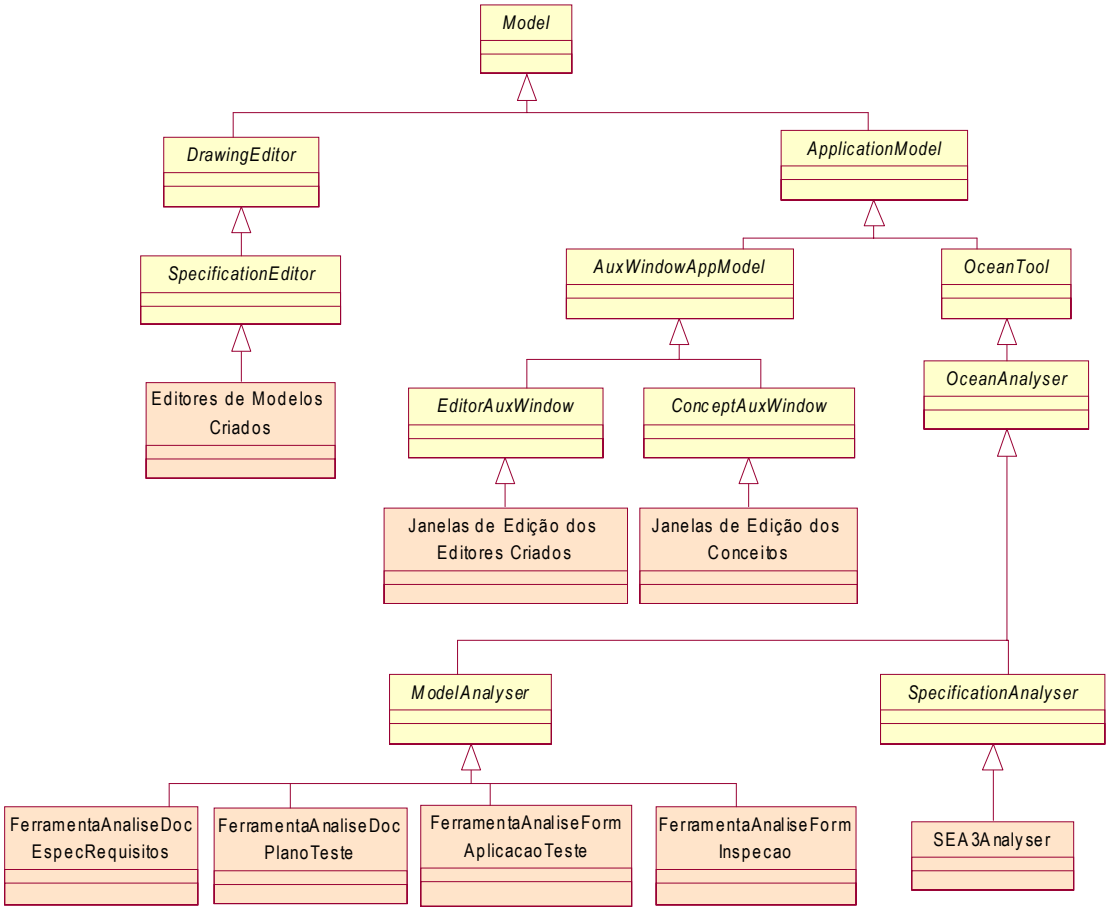


Figura 7.22 - Estrutura das ferramentas de análise

Cada ferramenta referente ao seu respectivo modelo permite apenas analisar a consistência do documento corrente.

A verificação de consistência das informações de cada modelo, é realizada através das ferramentas de cada um. Estas ferramentas obedecem a regras de consistência e percorrem todas elas para analisar a coerência dos documentos. As regras dos seus respectivos conceitos pertencentes aos referidos documentos são descritas no capítulo 6.

A ferramenta de análise da especificação analisa a consistência de todos os modelos da especificação corrente fornecendo um resultado, assim como a análise realizada pela ferramenta de cada modelo.

A análise de consistência dos documentos realizada pelas ferramentas é armazenada num arquivo tipo texto, num diretório definido após a instalação do ambiente SEA. Apenas deve-se informar um nome para o arquivo. Estas ferramentas incluem regras de consistência aos documentos criados, o que num documento doc, por exemplo, não acontece.

Para analisar o documento corrente deve-se proceder da seguinte forma: No editor do modelo atual, deve-se escolher a opção *Tools* da barra de menus, e escolher *load tool*...; Até o último procedimento executado aparecerá a seguinte janela, conforme apresenta a figura 7.23. Em seguida, selecionar a ferramenta disponível e clicar no botão *Load Tool* (ao lado do botão *Cancel*). Após esse segundo passo abrirá uma outra janela, conforme demostrada na figura 7.24. Ao escolher a opção *Analyse current model* na figura 7.25, uma outra janela será aberta, pedindo para informar o nome do arquivo onde será armazenado o resultado da análise realizada pela ferramenta do modelo, como apresenta a figura 7.25.

O nome do arquivo onde é armazenada cada resultado de análise de consistência é definido pelo próprio usuário do ambiente SEA, quando proceder tal ação. O diretório onde são armazenados os arquivos com o resultado da análise é criado após a instalação do ambiente SEA.

As janelas de edição das figuras 7.24 e 7.25 são padrão e, por isso, são iguais para todas as ferramentas de análise. O que modifica é a análise de consistência, pois cada modelo possui suas próprias regras a serem executadas em cada ferramenta.

Figura 7.23 - Janela da Ferramenta de Análise do Documento de Especificação de Requisitos

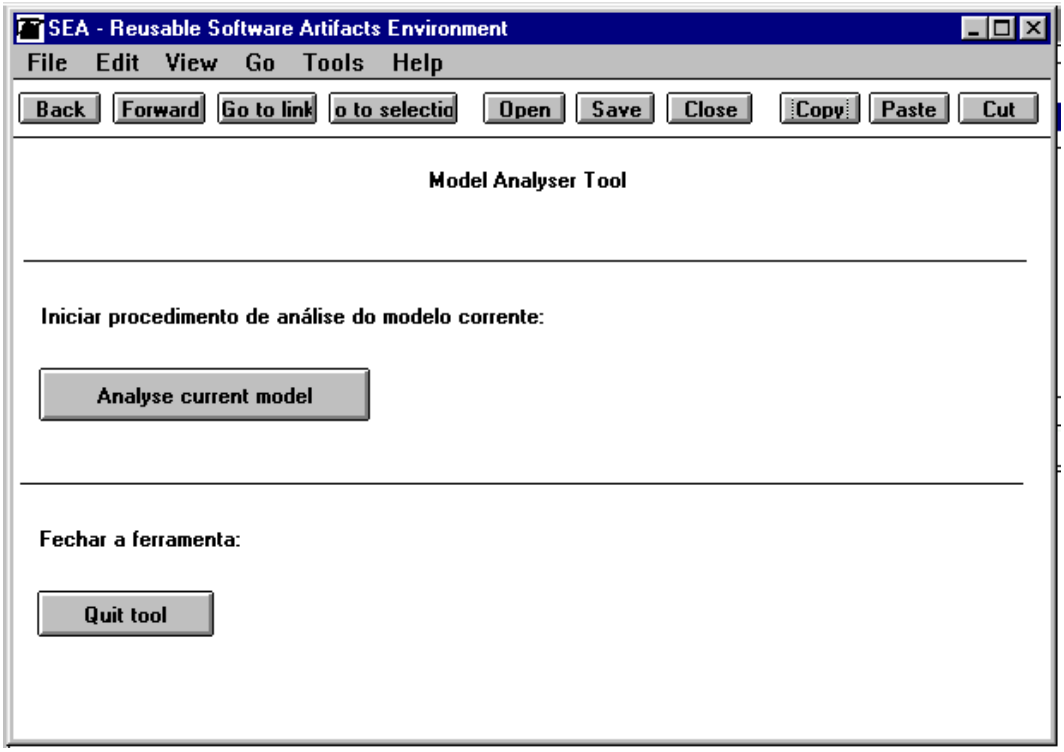


Figura 7.24 - Demonstração da janela da ferramenta de análise de consistência.

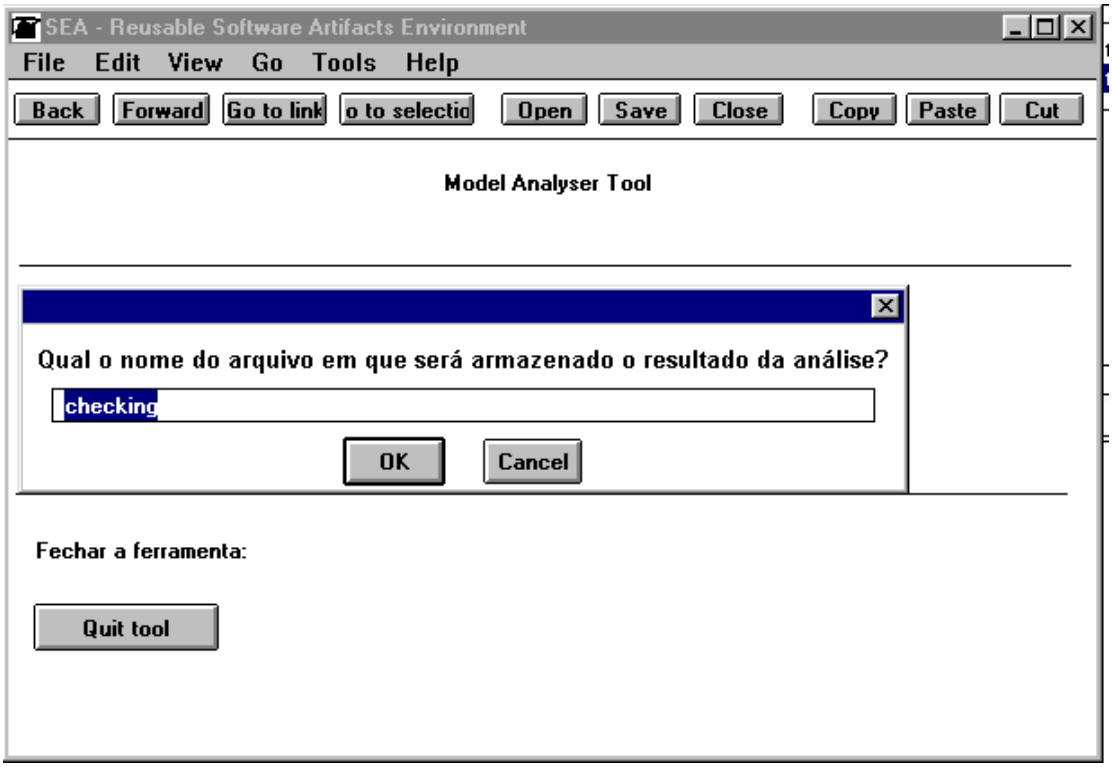


Figura 7.25 - Demonstração da janela em que deve ser informado o nome do arquivo que armazena o resultado da análise de consistência

Para cada modelo criado, existe uma ferramenta de análise de consistência associada. As ferramentas percorrem todas as regras que cada modelo possui. Essas restrições são convertidas em forma de regras e implementadas através da linguagem *Smalltalk*.

7.7 Consistência dos Documentos no Ambiente SEA

As ferramentas de análise de consistência verificam a consistência dos documentos criados. Cada ferramenta segue um conjunto de regras pertencentes a cada modelo, que devem ser obedecidas. Cada modelo possui suas regras específicas. As regras de consistência para cada modelo são descritas no capítulo 6.

Um documento (modelo) só é consistente se, todas as regras de consistência definidas no capítulo 6, forem seguidas. Caso pelo menos uma das regras não for obedecida, o documento não estará totalmente consistente.

A análise do documento é realizada internamente da seguinte forma: Primeiramente verifica-se se o modelo está vazio (apenas com o nome), caso seja falso, a ferramenta entra no método que percorre todas as regras de consistência dos conceitos daquele modelo. Após ter seguido esses passos, está concluída a verificação de consistência do modelo corrente. Para maiores detalhes das regras de consistência, verificar capítulo 6.

A verificação de consistência dos documentos é um complemento aos modelos criados no ambiente SEA.

7.8 Conclusão

A carência identificada no ambiente SEA, falta de registro de informações textuais motivou o desenvolvimento de uma estrutura de documentos textuais. Essa estrutura serviu como um ensaio do uso de documentos textuais no ambiente, sendo que apenas quatro documentos estão disponíveis, mas isso não impede que outros tipos de documentos textuais sejam adicionados a essa estrutura.

Para isso, primeiramente foi introduzida uma nova estrutura de especificação, Estrutura de Documentos Textuais que, entre outras aptidões, agrega quatro modelos de documentos. Esses modelos são *templates* contendo informações definidas como necessárias nos documentos disponíveis. Os modelos que fazem parte dessa estrutura são: Documento de Especificação de Requisitos, Documento de Plano de Teste, Formulário de Aplicação de Teste e Formulário de Inspeção.

Após a identificação dos modelos que fazem parte da nova estrutura de especificação, conceitos foram estabelecidos e definidos, os quais são referenciados por um ou mais modelos. Ressalta-se que cada documento é tratado como modelo e que cada seção de um documento é tratado como conceito, e ambos são elementos de especificação. Nesse contexto, fica visível a abordagem de reuso, pois um conceito pode ser usado por mais de um modelo.

Portanto, todas as características de determinado conceito são reutilizadas, da mesma forma, por documentos diferentes.

Definidos quais conceitos seriam referenciados pelos modelos, foram criados os editores de modelos. Os editores permitem a visualização de elementos de especificação no ambiente.

Com a permissão de visualizar todos os modelos e conceitos, janelas de edição dos conceitos disponíveis foram criadas e referenciadas pelos conceitos. Nas janelas de edição é permitido definir as informações referentes a cada conceito.

A definição dos modelos, conceitos e informações a serem estabelecidos na estrutura de documentos textuais seria suficiente para uma simples estrutura de documentos textuais estruturados, como contribuição parcial da proposta. Nesse sentido, o objetivo da documentação é, além de suprir a deficiência no ambiente SEA, fornecer subsídios para que os processos possam ser compatíveis com requisitos de CMMI *Staged* que necessitam de suporte à documentação.

As regras de consistência das informações presentes nos modelos são percorridas pelas ferramentas de análise de consistência que realizam essa verificação. Cada modelo possui uma ferramenta de análise de consistência do mesmo. O resultado dessa análise é armazenado num arquivo texto, fornecendo informações relevantes sobre a consistência dos modelos criados e analisados.

Todos os passos citados anteriormente foram descritos detalhamente neste capítulo em conjunto com uma representação gráfica, apresentando passo a passo como foi introduzida a nova estrutura de especificação ao ambiente SEA.

Essa nova estrutura de especificação no ambiente SEA ajuda na melhoria dos processos, aumentando a qualidade dos mesmos com o auxilio do modelo CMMI *Staged*.

8 Conclusão

A nova estrutura de especificação introduzida ao ambiente SEA, é caracterizada por um conjunto de elementos de especificação (modelos e conceitos). Essa estrutura foi criada por meio do *framework* OCEAN (SILVA, 2000), que entre outras características, proporciona subsídios para a construção de novas estruturas de especificação, bem como o suporte à criação de elementos de especificação.

Estrutura de Documentos Textuais (nova estrutura de especificação) possui a finalidade de suprir uma deficiência identificada no ambiente: falta de suporte ao registro de informações textuais.

Essa carência identificada no SEA e parcialmente solucionada (a nova estrutura não abrange todos os possíveis tipos de documentos) foi devido ao Projeto de *Workflow*, estrutura de especificação desenvolvida por (SCHEIDT, 2002) necessitar de uma estrutura de documentação que fornecesse um conjunto de documentos para documentar atividades que precisam de registro textual. O Projeto de *Workflow*, permite que atividades a serem realizadas durante um projeto sejam definidas. Por exemplo, ao estabelecer no Projeto de *Workflow*, uma atividade de inspeção ou um plano de teste, o ambiente SEA não oferecia até então, nenhum tipo de estrutura para que fosse possível registrar as informações referentes a atividades desse tipo. Apenas era definido como "*frozen* - atividade congelada", quando uma atividade de registro textual era concluída. Essa definição é fornecida na própria estrutura de especificação - Projeto de *Workflow*.

Assim, a estrutura de documentos textuais disponibiliza quatro documentos (modelos) estruturados: Documento de Especificação de Requisitos, Documento de Plano de Teste, Formulário de Aplicação de Teste e Formulário de Inspeção. Esses documentos textuais proporcionam informações a serem definidas, identificadas como essenciais e algumas como necessárias em cada documento. Nesses documentos, é permitido incluir, editar e deletar informações.

A seguir, são apresentados os resultados obtidos e limitações, bem como as sugestões para trabalhos futuros.

8.1 Resultados Obtidos

O trabalho de pesquisa desenvolvido produziu resultados identificados como positivos, pois acredita-se ter solucionado parte dos problemas. As contribuições acrescentadas ao ambiente SEA são descritas abaixo.

- Foi proposta uma estrutura de documentos textuais ao ambiente SEA. Essa estrutura de documentos, inserida ao ambiente é caracterizada por um conjunto de quatro documentos textuais estruturados, tratados como modelos. A estrutura criada foi construída a partir das funcionalidades fornecidas pelo *framework* OCEAN. Tem a finalidade de proporcionar ao ambiente, melhor organização das informações textuais e o registro das mesmas. Com isso, insere-se a uma ferramenta CASE, mais qualidade em seus processos;

- Tratamento organizado dos documentos textuais estruturados. Este tratamento é estabelecido por meio de regras de consistência (capítulo 6) definidas na própria especificação. As regras de consistência são restrições tratadas nos modelos. São regras que definem quais são as informações essenciais para que cada modelo (documento) mantenha sua consistência;

- Ferramentas de consistência responsáveis por realizar a análise de consistência de cada modelo. Essas ferramentas percorrem todas as regras definidas nos modelos para realizar a análise de consistência do modelo e/ou da especificação corrente;

- Armazenamento da análise de consistência num arquivo texto. Essa análise é disponibilizada para que possa ter um acompanhamento das informações dos modelos e para a identificação das possíveis alterações que devam ocorrer para que se obtenha uma especificação consistente;

- Os conceitos definidos na nova estrutura de especificação podem ser reutilizados por outros documentos que possam vir a ser adicionados a essa estrutura, pois existem conceitos como identificador do documento, histórico do documento,

agendamento que podem perfeitamente ser reutilizados por outros documentos textuais. Isso já acontece nessa estrutura, como, por exemplo, o documento de especificação de requisitos e o documento plano de teste referenciam os conceitos identificação do documento, introdução e histórico do documento.

8.2 Limitações

Contribuições ao ambiente SEA foram acrescentadas e ainda, o ambiente possui algumas limitações:

- Não foram introduzidos ao ambiente SEA todos os documentos necessários para registro das informações referentes às atividades ao longo do processo de desenvolvimento de *software*, contemplados ao CMMI.

8.3 Trabalhos Futuros

Conforme descrito nos capítulos anteriores, a possibilidade de extensão do *framework* OCEAN é considerada acentuada devido à sua flexibilidade. Considerando essa afirmação, ressalta-se que o ambiente SEA, assim como outro ambiente de desenvolvimento construído a partir do OCEAN poderá receber diversas funcionalidades e adaptações para melhorar a eficiência e qualidade nos seus processos. Essas funcionalidades podem ser criadas a partir dos recursos oferecidos pelo *framework*.

A nova estrutura de especificação criada com o apoio do OCEAN e adaptada no ambiente SEA comprova tal flexibilidade. Como a estrutura de documentos textuais implementada é um protótipo, nem todos os documentos textuais previstos num projeto, que obedeça algum modelo de qualidade no processo, foram adicionados. Assim, segue algumas sugestões para trabalhos futuros que possam melhorar a estrutura criada, e os recursos oferecidos pelo ambiente SEA, permitindo a evolução do mesmo por meio do *framework* OCEAN.

- **Propor novos tipos de documentos textuais à estrutura criada:** A adaptação de novos documentos proporcionará ao ambiente SEA uma variedade de opções relacionadas ao registro de atividades que demandam documentação.

- **Matriz de Rastreabilidade:** Incluir um novo documento com a finalidade de uma matriz de rastreabilidade. Essa matriz será responsável por informar se requisitos definidos no documento de especificação de requisitos estão sendo aplicados por algum projeto e se existe algum teste para determinado requisito;

- **Áreas de processo para auxiliar na melhoria de qualidade no processo de desenvolvimento:** Identificar e implantar PAs de CMMI *Staged* que possam ser adaptadas ao ambiente SEA.

8.4 Considerações Finais

As atividades aplicadas ao processo de desenvolvimento de *software* têm a finalidade de melhorar a qualidade nos seus processos, mas, apesar disso, não são práticas adotadas normalmente por organizações. Por outro lado, melhorar a qualidade no processo de desenvolvimento de *software* tem sido uma preocupação constante nas organizações que pretendem adotar um modelo de qualidade nos processos e que almejam a qualidade de seus produtos finais.

Nesse contexto, o trabalho foi motivado pelas vantagens em adaptar práticas baseadas no modelo CMMI *Staged* ao ambiente SEA, com a finalidade de melhorar a qualidade dos processos em termos de produtos de trabalho dentro de uma ferramenta CASE, atingindo assim seu objetivo.

9 Referências Bibliográficas

BRANDT, J. ***HotDraw***. Urbana: University of Illinois at Urbana-Champaign, 1995. Master thesis.

CORDENONZI, Walkíria Helena. **Mapeamento de Padrões Internacionais de Qualidade de Produto e de *Software* para um Modelo Conceitual de Gerência do Processo de Desenvolvimento de *Software***. Dissertação de Mestrado. Instituto de Informática: Universidade Federal do Rio Grande do Sul. Porto Alegre, RS, Janeiro de 2000.

DAMIAN, Adtrian. ***Software Frameworks***. University of Calgary. Março de 1998. Disponível em: http://sern.ucalgary.ca/courses/SENG/609.03/W98/adi2/, 05 de Novembro de 2002.

CORDEIRO, Marco Aurélio. **Uma Ferramenta Automatizada de Suporte ao Processo de Gerenciamento de Requisitos**. Dissertação de Mestrado. Pontifícia Universidade Católica do Paraná. Curitiba, PR, 2002. Disponível em http://www.pucpr.br/teses, 28 de Abril de 2003.

FELTRIN, Valéria Delisandra. **Apoio à Documentação de Engenharia Reversa de *Software* por meio de Hipertextos**. Dissertação de Mestrado. Instituto de Ciências Matemáticas e de Computação - Universidade de São Paulo. São Paulo, SC, Dezembro de 1999. Disponível em: http://www.teses.usp.br/teses, 28 de Abril de 2003.

FIORINI, Soeli; LEITE, Julio C. S.; LUCENA, Carlos J. P. **Organizando Processos de Requisitos**. Pontifícia Universidade Católica do Rio de Janeiro - PUC-Rio. Publicado no Workshop de Engenharia de Requisitos 1998 (WER' 98). Maringá - PR, 98. Disponível em http://www.inf.puc-rio.br/~wer98, 02 de Abril de 2003.

HOLLINGSWORTH, David. ***Workflow Management Coalition***. The *Workflow* Reference Model, 1995. Disponível em http://www.wfmc.org, 25 de Junho de 2002.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. **ISO/IEC 15504**. Disponível em www.iso.ch, 10 de Abril de 2004a.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. ISO 9001. Disponível em www.iso.ch, 10 de Abril de 2004b.

JOHNSON, Ralph E., Components, Frameworks, Patterns. Fevereiro, 1997. Communications of the ACM, Outubro de 1997.

KRUCHTEN, Philippe. ***Rational Unified Process***. Rational Software, 2001. Canadá. Disponível em http://www.therationaledge.com/content/jan_01/f_rup_pk.html, 10 de Janeiro de 2004.

LIMA, Carla Alessandra Gomes; REIS, Rodrigo Quites; NUNES, Daltro José. **Gerenciamento do Processo de Desenvolvimento Cooperativo de *Software* no Ambiente PROSOFT**. Departamento de Informática - Universidade Federal do Pará, Belém, PA. Publicação XII Simpósio Brasileiro de Engenharia de *Software* - SBES, 1998. Disponível em http://www.inf.ufrgs.br/~clima/artigos/prosoft/sbes98.html, 16 de Março de 2003.

MANZONI, Lizandra Vielmo. **Sistema de Gerência de *Workflow* para Apoiar o Desenvolvimento de *Software* baseado no Processo Unificado da *Rational***. Dissertação de Mestrado. PPGC - Universidade Federal do Rio Grande do Sul. Porto Alegre, RS, 2001. Disponível em http://www.ufrgs.br/, 25 de Abril de 2003.

HERNDON, Mary Ane. MOORE, Robert. PHILLIPS, Mike. WALKER, Julie. WEST, Laura. ***Interpreting Capability Maturity Model Integration (CMMI) for Service Organizations - a Systems Engineering and Integration Services Examples***. Carnegie Mellon University, Novembro de 2003. Disponível e publicado em: http://www.sei.cmu.edu/pub/documents/03.reports/pdf/03sr009-revised.pdf. Acesso em 15 de Janeiro de 2004.

MIAN, P. G.; NATALI, A. C. C.; FALBO, R. A. **Ambientes de Desenvolvimento de *Software* e o Projeto ADS**. Laboratório de Engenharia de *Software* - Departamento de Informática, Universidade Federal do Espírito Santo, Vitória, ES, 1999. Disponível em http://www.inf.ufes.br/~falbo/download/pub/RevistaCT072001.pdf, 08 de Junho de 2003.

OBJECT MANAGEMENT GROUP. ***Introduction To OMG´s - Unified Modeling Language***. Disponível em: http://www.omg.org, 10 de Dezembro de 2003.

SANTANDER, Victor F. A.; VASCONCELOS, Alexandre M. L.. **Mapeando o Processo Unificado em Relação ao CMM - Nível 2**. Centro de Informática. Universidade Federal de Pernambuco, Recife - PE. Disponível em http://www.qualiti.com.br/artigos.html, 19 de Abril de 2002.

SCHEIDT, Neiva. **Estudo e Aplicação de *Workflow* e Requisitos de CMM em um Ambiente de Desenvolvimento de *Software***. Dissertação de Mestrado. PGCC - Universidade Federal de Santa Catarina - UFSC. Florianópolis, SC, Dezembro de 2002.

SILVA, Ricardo Pereira e. **Suporte ao Desenvolvimento e uso de *Frameworks* e Componentes**. Tese de Doutorado. Instituto de Informática da Universidade Federal do Rio Grande do Sul - UFRGS. Porto Alegre, RS. 2000.

SILVA, Ricardo Pereira e. PRICE, Tom. ***Tool Suport for Helping the Use of Frameworks***. In: Proceddings of XIII Internacional Conference of the Chilean Computer Science Society (SCCC´98). Chile, 1998a.

SILVA, Ricardo Pereira e. PRICE, Tom. **A Busca da Generalidade, Flexibilidade e Extensibilidade no Processo de Desenvolvimento de *Frameworks* Orientados a Objetos.** . In: Proceddings of Workshop Iberoamericano de Engenharia de Requisitos e Ambientes de Software (IDEAS, 98) Torres, 1998b.

SOFTWARE ENGINEERING INSTITUTE. ***Capability Maturity Model® for Software (SW-CMM®)***. *Carnegie Mellon University*. Disponível em: http://www.sei.cmu.edu/publications/documents/94.reports/94.tr.012.html, 15 de Janeiro de 2002.

SOFTWARE ENGINEERING INSTITUTE. ***CMMI^{SM} for Systems Engineering/Software Engineering/Integrated Product and Process Development, Version 1.1, Staged Representation (CMMI-SE/SW/IPPD, V1.1, Staged).*** *Carnegie Mellon University,* Agosto

de 2003a. Disponível em: http://www.sei.cmu.edu/publications/documents/02.reports/02tr001.html, Setembro de 2003.

SOFTWARE ENGINEERING INSTITUTE. ***CMMI^{SM} for Systems Engineering/Software Engineering, Version 1.1, Continuous Representation (CMMI-SE/SW, V1.1, Continuous).*** *Carnegie Mellon University,* Agosto de 2003b. Disponível em: http://www.sei.cmu.edu/publications/documents/02.reports/02tr001.html, Setembro de 2003.

TALIGENT, Inc. ***Building Object-Oriented Frameworks.*** 1997. Disponível por http://lhcb-comp.web.cern.ch/lhcb-comp/Components/postscript/buildingoo.pdf, 15 de Outubro de 2002.

TRAVASSOS, G. **O Modelo de Integração de Ferramentas da Estação TABA**. Tese de Doutorado. Sc. COPPE, Universidade Federal do Rio de Janeiro - UFRJ. Rio de Janeiro - RJ. 1994.

ZSCHORNACK, Fabio. **Evolução de Esquemas de *Workflow* Representados em XML**. Dissertação de Mestrado. Instituto de Informática da Universidade Federal do Rio Grande do Sul - UFRGS. Porto Alegre, RS. Abril de 2003. Disponível em http://www.ufrgs.br/, 15 de Janeiro de 2004.

WORKFLOW MANAGEMENT COALITION. ***Workflow*** **Reference Model**. Bruxelas, Fevereiro de 1999. Disponível em http://www.wfmc.org, 20 de Dezembro de 2002.

Apêndice

Exemplo no Ambiente SEA utilizando a Especificação de Estrutura de Documentos Textuais

Neste exemplo será apresentada uma situação em que serão definidas no Projeto de *Workflow* (maiores detalhes no capítulo 3), atividades que necessitam de documentação textual estruturada. O Projeto de *Workflow* é modelado através de um Editor de *Workflow* (SCHEIDT, 2002).

Como mostra a figura 1, as atividades definidas no Projeto de *Workflow* são: definir requisitos de um componente (requisitos componente); inspecionar os requisitos estabelecidos (inspecionar requisitos); criar um *framework* para interface (*framework* interface); especificar interface; inspecionar interface; especificar componente; inspecionar componente.

De acordo com (SCHEIDT, 2002) as atividades devem ser definidas como humanas ou automatizadas.

Cada atividade possui uma janela para identificação, definir seus atributos e para definir um *link* que pode estar associado a uma especificação ou documento. Na janela de edição para definir os atributos de cada atividade deve se dizer se a atividade é automatizada ou humana. O *link* é obrigatório nas atividades automatizadas.

A figura 1 apresenta um Projeto de *Workflow* organizado através do seu editor.

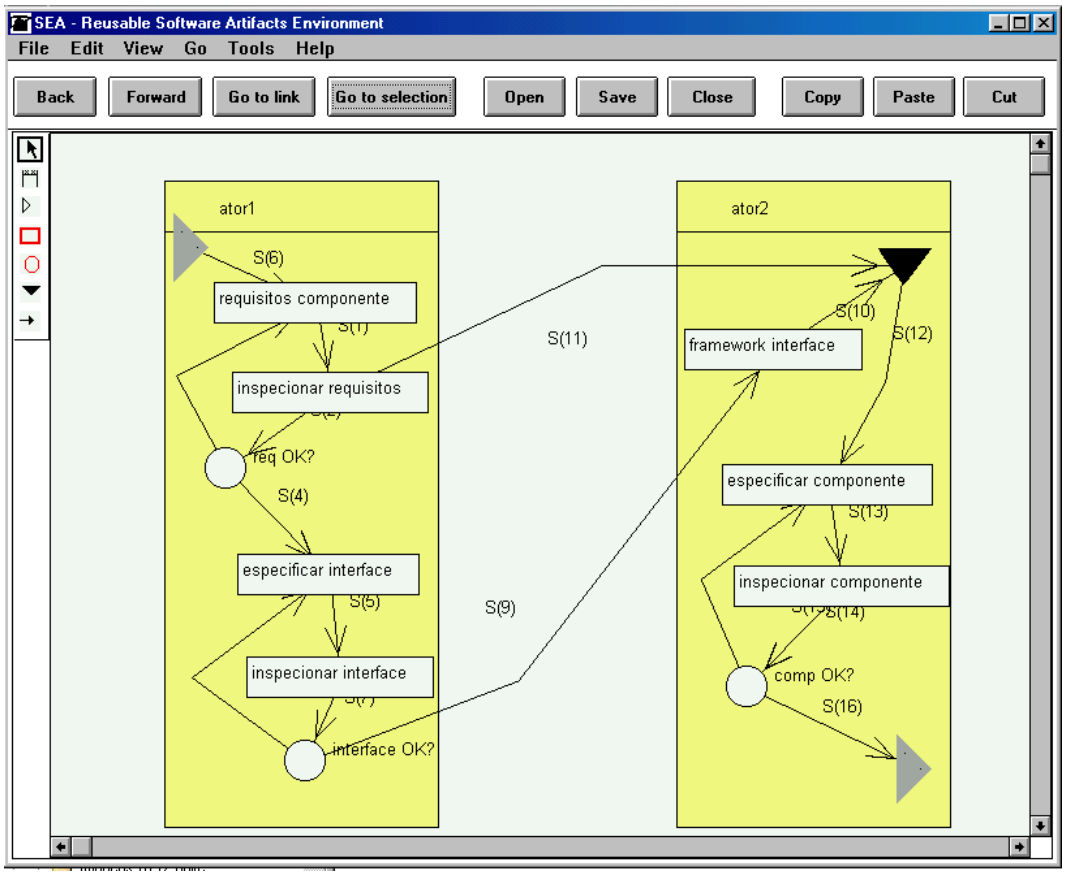


Figura 1 – Projeto de *Workflow*

As atividades a serem realizadas são controladas pelo sistema gerenciador de *Workflow*. O sistema gerenciador é uma ferramenta que verifica e gerencia as atividades e processos a serem realizados. O ambiente SEA permite que essa ferramenta seja acionada caso a especificação de *Workflow* esteja "*Frozen*", isto é, congelada. Essa opção só é permitida caso a especificação esteja validada, ou seja, caso a especificação esteja "ok". Assim, apenas especificações de *Workflow* consistentes podem ser gerenciadas.

As atividades a serem realizadas podem estar dentro de um dos três estados: "*frozen - congelado*", "*not frozen – não congelado*", "sem documento associado ou documento vazio" ou "*not frozen – não congelado* e atividade processada". O estado de cada atividade é representado no gerenciador de *workflow* e tal estado depende do documento que esta está associada. Os estados, dos quais um deles é definido para tal atividade, são disponibilizados no próprio ambiente SEA.

No caso de necessitar de uma decisão entre quais atividades devem ser processadas e quais atividades devem ser rejeitadas, as não escolhidas pela decisão recebem o estado de descartada. No caso das atividades humanas e automatizadas estarem no estado "não processada", elas aparecem na página onde se encontra a opção do sistema gerenciador de *Workflow*. As atividades humanas são consideradas processadas, apenas quando o ator responsável por tal atividade, as seleciona e clica no botão "*Performed*". Em relação às atividades automatizadas, elas apenas são representadas na página com seus atributos (ver capítulo 3).

Todas as atividades citadas necessitam ser concluídas e, para que uma nova atividade seja concluída, ou seja, congelada, a anterior deve estar congelada também.

Algumas atividades definidas no Projeto de *Workflow*, podem ser congeladas automaticamente, desde que os procedimentos necessários para que isso aconteça sejam obedecidos.

Para que no Projeto de *Workflow* as atividades sejam definidas de acordo com o estado que estão, nas suas respectivas especificações deve ter sido definido o seu estado para que o gerenciador de *workflow* possa processar o seu estado também. Isto vale para todas as atividades definidas num Projeto de *Workflow*.

Os estados das atividades são controlados pelo gerenciador de *workflow* através de cores que identificam o estado das mesmas. Por exemplo, a cor branca representa uma atividade ainda não processada; a azul, uma atividade em andamento; a verde, uma atividade já processada, e a cor vermelha, uma atividade irregular.

Através da figura 1 serão demonstrados os passos do processamento de cada atividade. Como se pôde observar nessa figura, todas as atividades ainda não foram processadas, pois apresentam a cor "branca".

Na figura 2 é apresentado o estado da atividade especificar componente que, de acordo com a cor definida, a especificação que permite criar componente, o qual essa atividade está apontando (através de um *link*) já foi validado, bem como a atividade já foi analisada e

validada. Com isso, a atividade se encontra no estado de processada, isto é, "*frozen* – congelada".

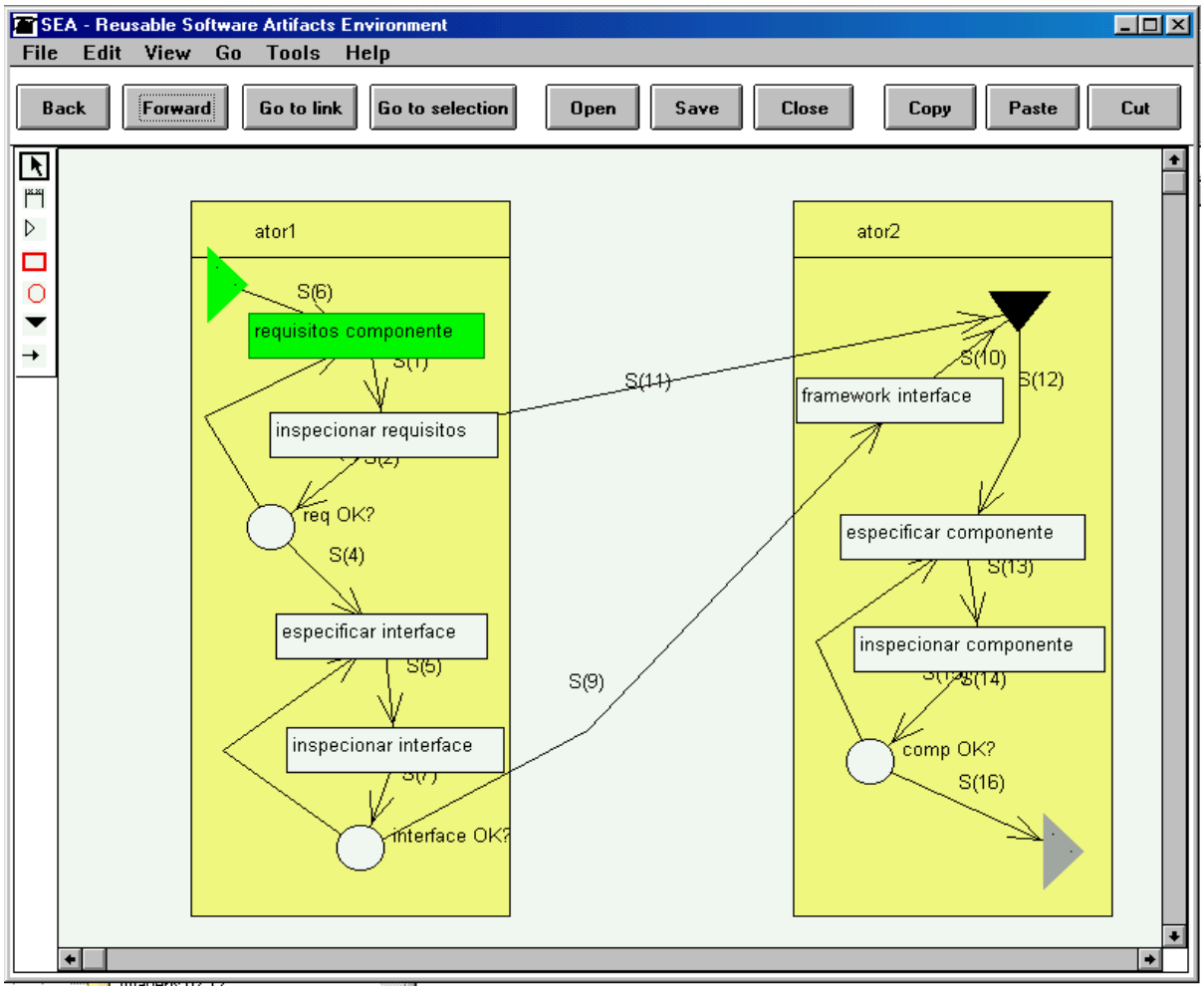


Figura 2 – Atividade requisitos componente, realizada

A figura 4 apresenta o estado da atividade inspecionar requisitos. A atividade inspecionar requisitos (figura4) se encontra na cor azul, ou seja, a atividade ainda está em andamento.

Numa versão mais antiga do ambiente SEA, a atividade inspecionar requisitos não poderia ser definida como automatizada, pois não existia uma ferramenta acoplada ao ambiente que fornecesse documentos textuais estruturados como é o caso do documento de inspeção. Com a inserção de uma nova especificação ao ambiente SEA, a ferramenta suporta quatro tipos de documentos textuais estruturados na especificação mencionada: documento de

especificação de requisitos, documento de plano de teste, formulário de aplicação de teste e formulário de inspeção.

Sem a nova especificação, essas atividades eram definidas como humanas, quando eram estabelecidas no Projeto de *Workflow*. Portanto, as informações desses documentos não eram controladas pelo gerenciador do *workflow*. Quando atividades que necessitam de documentos estruturados eram concluídas, uma pessoa definia como processada (concluída), ou seja, congelada.

Com a nova especificação - documentos textuais estruturados - quando cada atividade relacionada a um dos quatro documentos citados, for estabelecida no projeto de *workflow*, esta pode ser definida como automatizada, fazendo com que seja possível definir um *link* estabelecendo a que documento está associado aquela atividade e qual a especificação que pertence. A figura 3 apresenta a criação da nova especificação.

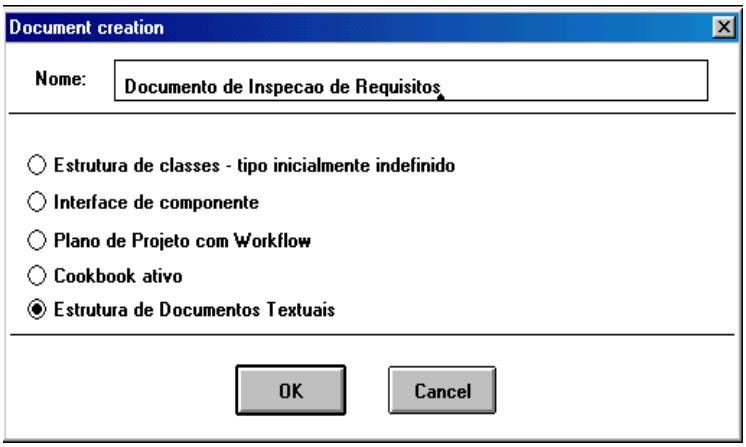


Figura 3 – Nova Estrutura de Documentos Textuais – Documento de Inspecao de Requisitos

Partindo dessa nova estruturação, a atividade inspecionar requisitos está sendo apontada pela especificação onde está definido o projeto de *workflow* (ou seja, onde é estabelecido as atividades a serem realizadas).

Para a atividade inspecionar requisitos (figura 4) estar sendo representada pela cor azul, o documento de inspeção responsável por armazenar informações sobre tal inspeção

ainda está em andamento, ou seja, foi definida como “atividade em andamento” na especificação Estrutura de Documentos Textuais. Essa definição acontece porque há campos ainda não preenchidos, por exemplo, o documento ainda não foi congelado e muito menos validado para verificação da consistência de suas informações.

O documento de inspeção apresentado na figura 5 demonstra as informações parcialmente preenchidas.

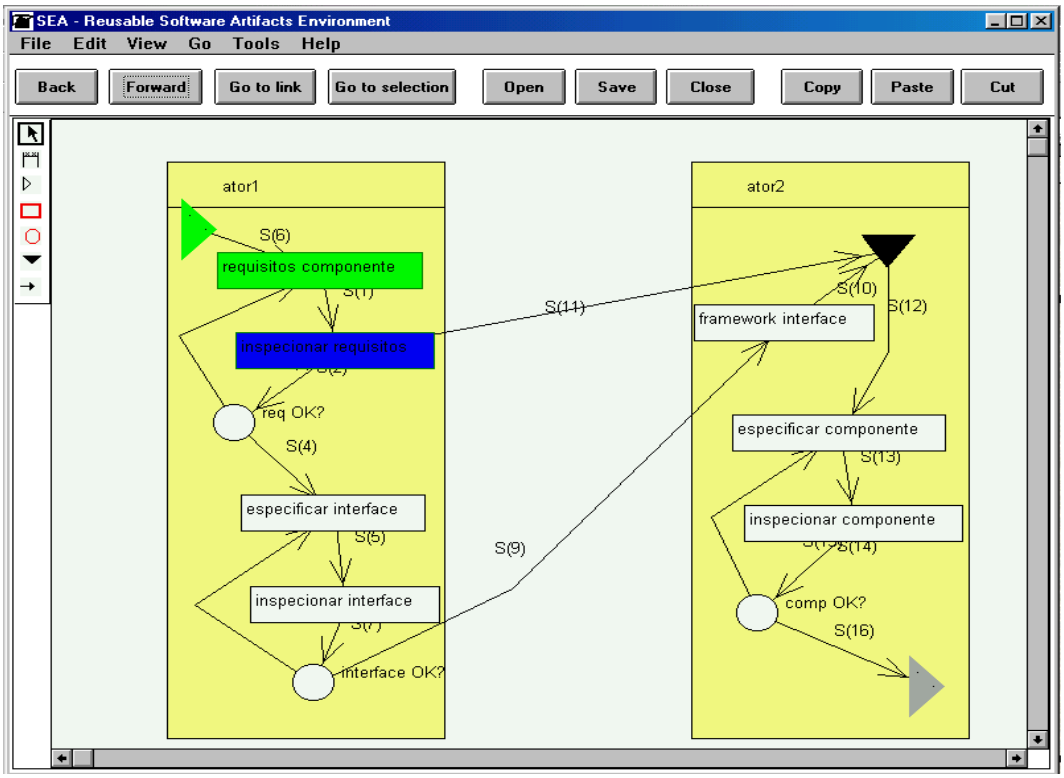


Figura 4 – Atividade inspecionar requisito, em andamento

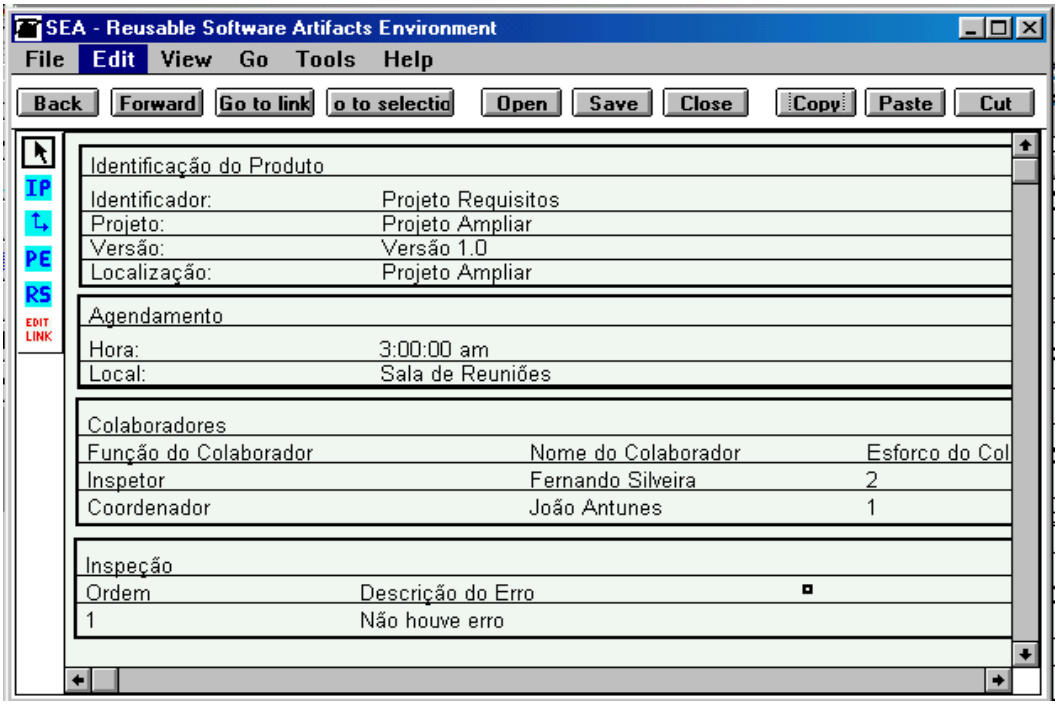


Figura 5 – Documento de Inspeção – atividade inspecionar requisitos, parcialmente preenchido

O documento apresentado pela figura 6 apresenta o documento todo preenchido, validado, verificada a consistência de suas informações através da ferramenta de análise de consistência e definido como *frozen*.

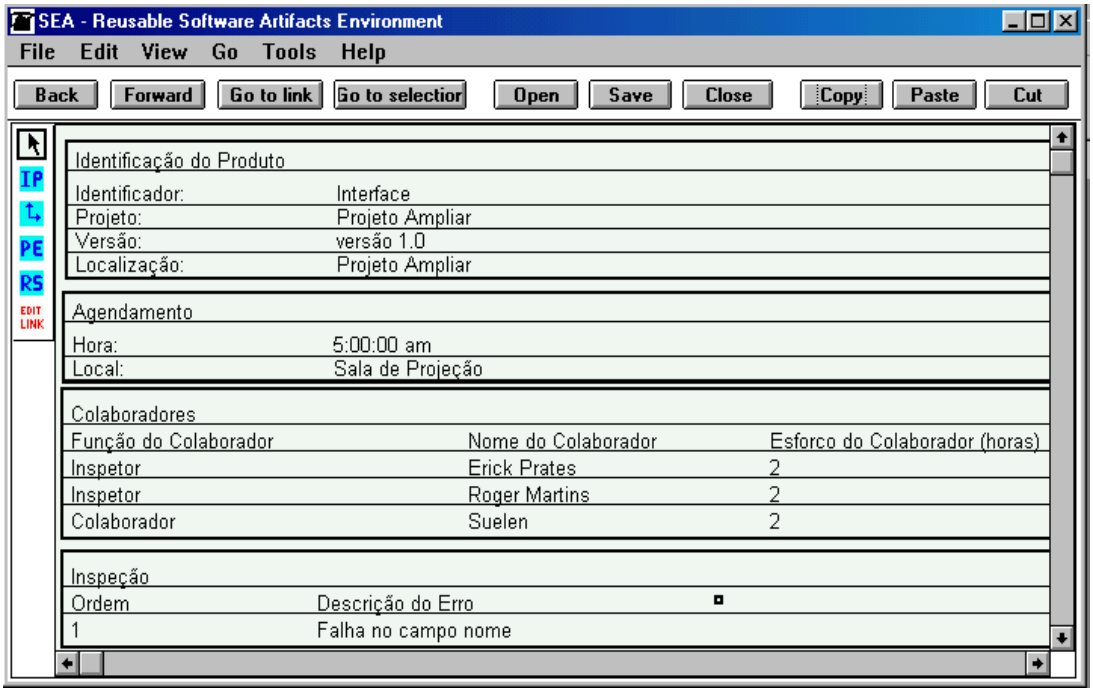


Figura 6 – Documento Formulário de Inspeção da Atividade Inspecionar requisito concluído

Após o preenchimento das informações (figura 6), o documento deve passar por uma verificação de consistência para ser validado e congelado. A figura 7 apresenta o início do processo de verificação de consistência.

Através opção *Tools* do menu são disponibilizados dois tipos de ferramentas de consistência, que para este exemplo deve ser selecionado a opção Ferramenta de Análise do Formulário de Inspeção como apresenta a figura 7.

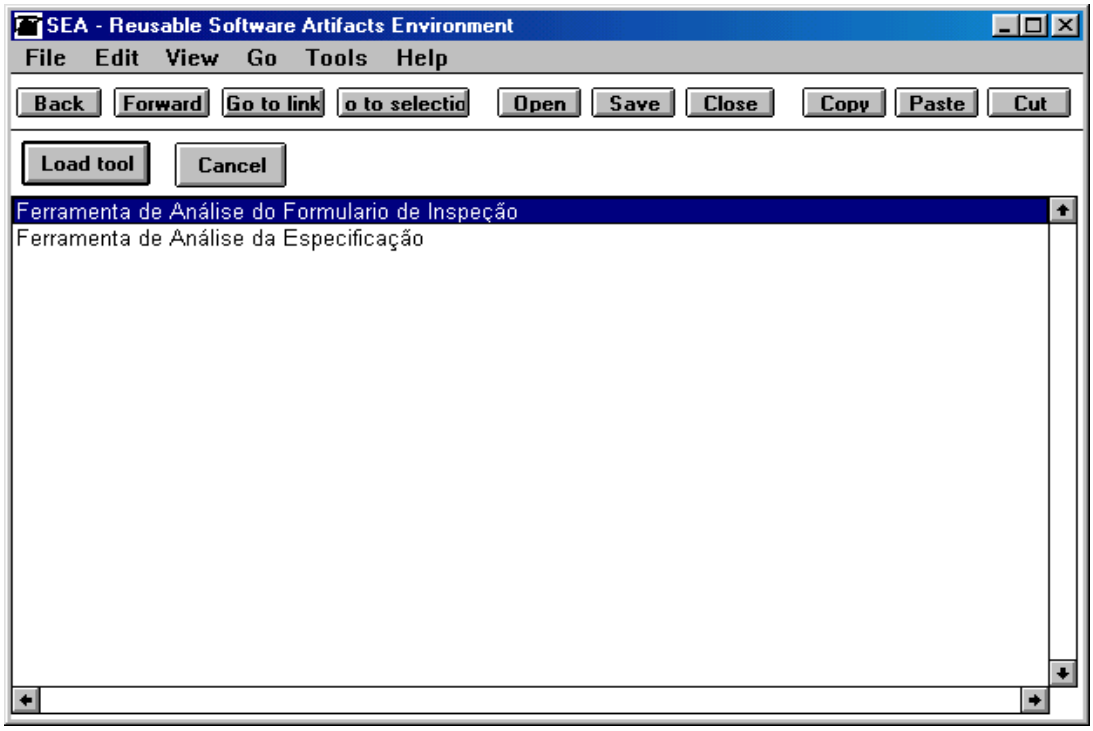


Figura 7 – Ferramentas de Análise de Consistência do documento formulário de inspeção

Seguindo o procedimento de análise de consistência, aparecerá a janela para selecionar a opção que realiza a análise de consistência – opção "*Analyse current model*", como apresenta a figura 8.

A janela de análise de consistência é a mesma para qualquer modelo, mas o procedimento em si executa restrições diferentes, dependendo do modelo, claro.

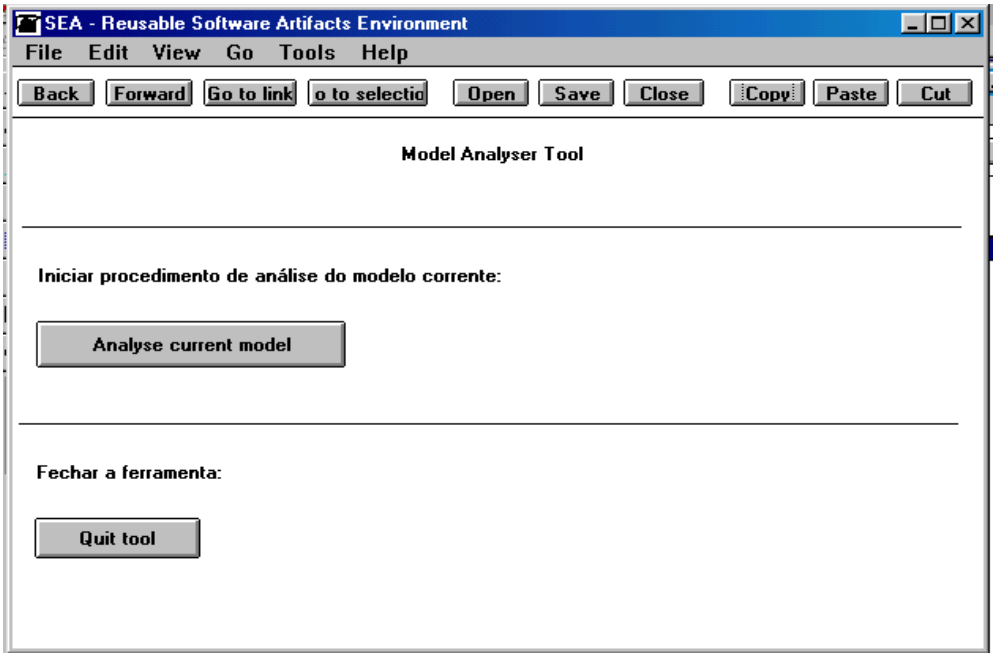


Figura 8 – Opção da Análise de Consistência do Modelo

Após a análise do documento ter sido identificado como "ok", sem nenhum erro, é necessário que o documento seja definido como *frozen* – congelado – ou seja, o documento está concluído, opção *freeze document* (*frozen*). A figura 9 apresenta a opção. As opções do menu permanecem em todas as especificações do ambiente SEA. Tal procedimento vale para qualquer situação de uma especificação.

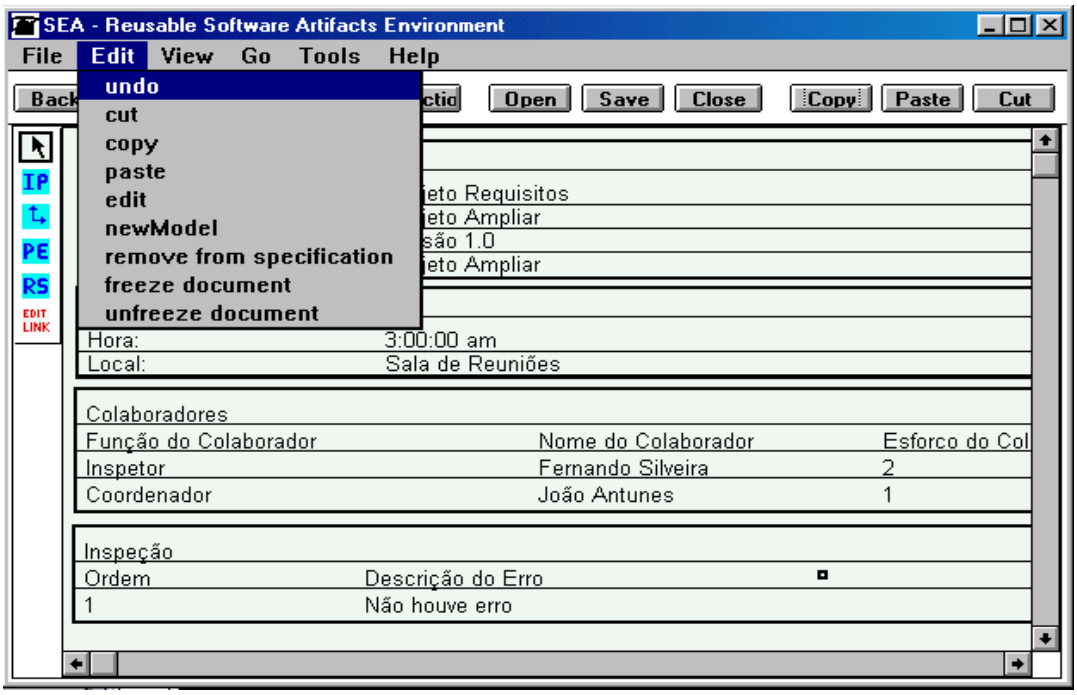


Figura 9 – Demonstração da Opção *freeze document*

Seguindo a demonstração do exemplo, a figura 10 apresenta a validação do documento inspecionar requisitos, definido na Estrutura de Documentos Textuais, pois a atividade inspecionar requisitos estando na cor verde, representa a validação e o "congelamento" – *frozen* das informações do documento criado e "linkado" pela referida atividade.

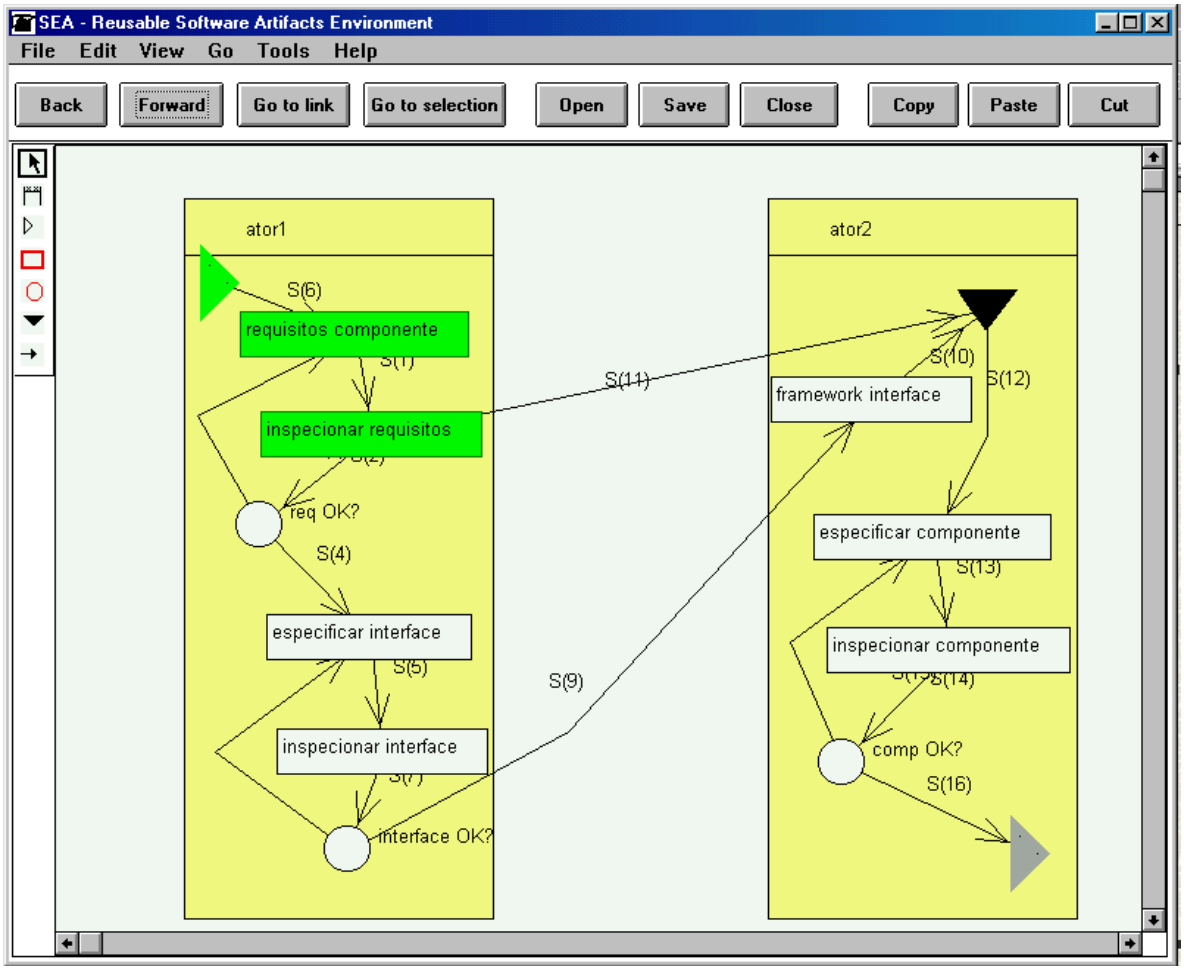


Figura 10 – Atividade inspecionar requisitos, realizada

Após a conclusão da atividade inspecionar requisitos, é possível tomar uma decisão como apresenta a figura 11 e passar para a próxima atividade definida no projeto de *workflow*.

Tomada a decisão, representada pelo circulo verde (figura 11), é possível começar a realizar a atividade especificar interface, a qual faz parte de outra especificação, que não a especificação Estrutura de Documentos Textuais com alguns documentos disponíveis.

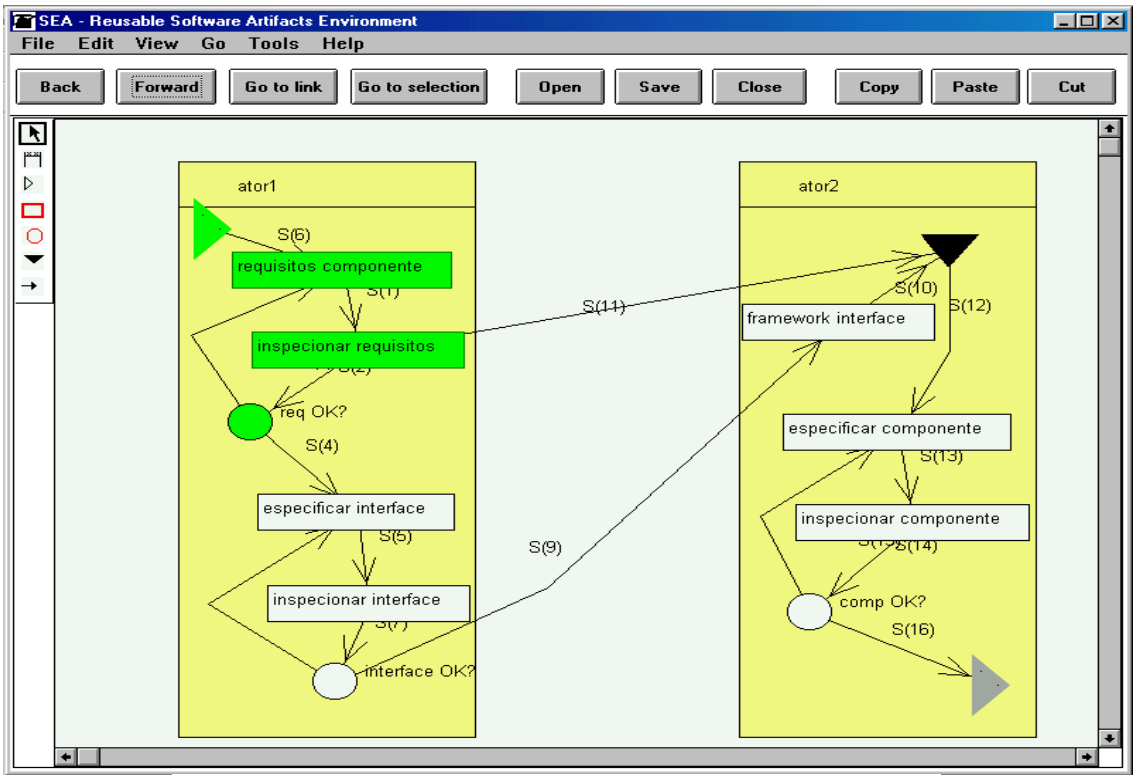


Figura 11 – Representação do símbolo para tomada de uma decisão

Após ter sido criada a especificação e o modelo da atividade especificar interface e, então, criado um *link* da atividade ao modelo que possui a realização da atividade definida, pode-se começar a concretização da referida atividade. A figura 12 apresenta atividade em andamento, isto é, a atividade ainda não foi concluída, nem passou por uma avaliação de consistência, apenas foi definida como “atividade em andamento”, pois está representada em azul.

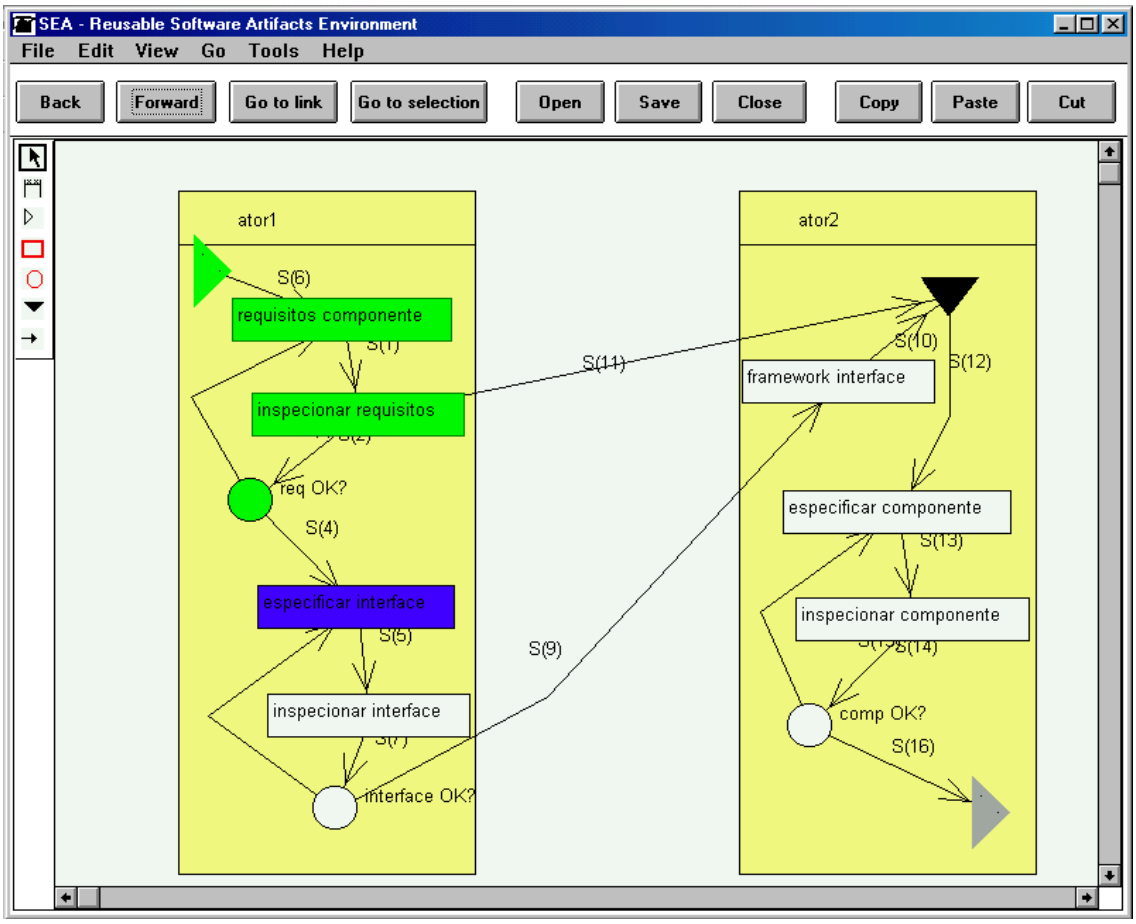


Figura 12 – Atividade especificar interface, em andamento.

A figura 13 apresenta a atividade especificar interface realizada, verificada a consistência e definida como *frozen*. A cor verde na atividade especificar interface, precede tal afirmação.

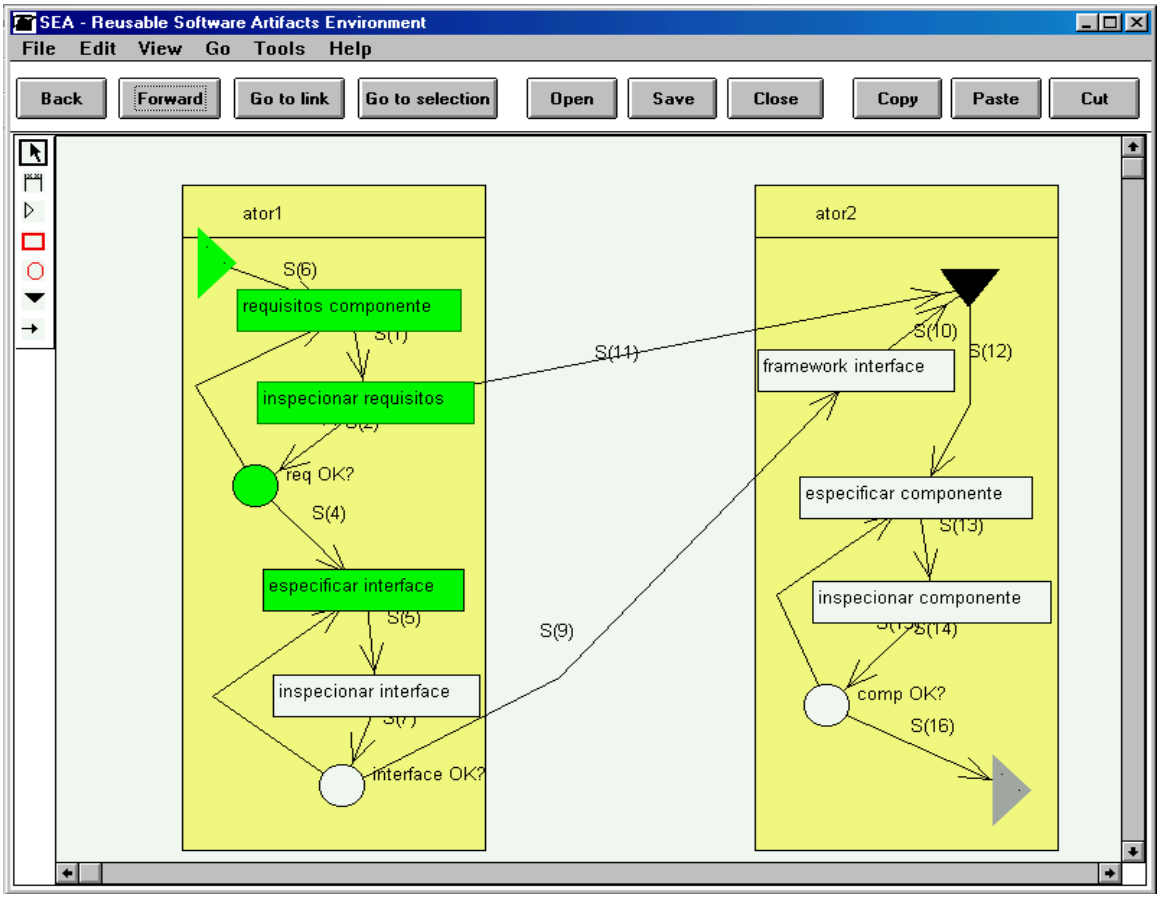


Figura 13 – Atividade especificar interface, realizada

Após a conclusão da atividade especificar requisitos, pode-se dar início a atividade inspecionar requisitos, na qual esta atividade pode ser linkada a um documento para inspecionar requisitos disponível na especificação Estrutura de Documentos Textuais.

A figura 14 apresenta a atividade em andamento, isto é, ainda não foi verificada sua consistência, pois suas informações ainda não estão consistente devido a não-informação de algum dado, por isso a cor azul da atividade.

A figura 15 apresenta um exemplo do documento inspecionar interface em andamento, ou seja, com algumas informações preenchidas e documento definido, em andamento.

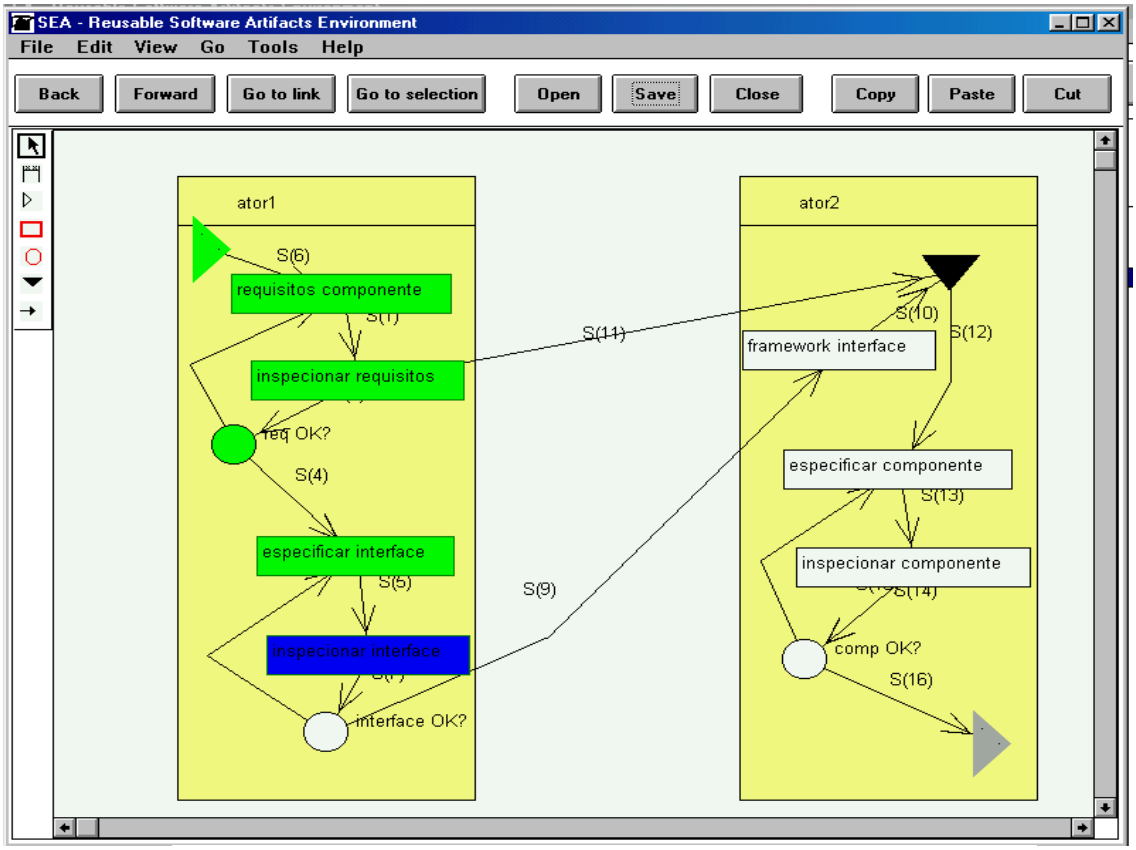


Figura 14 – Atividade inspecionar interface, em andamento

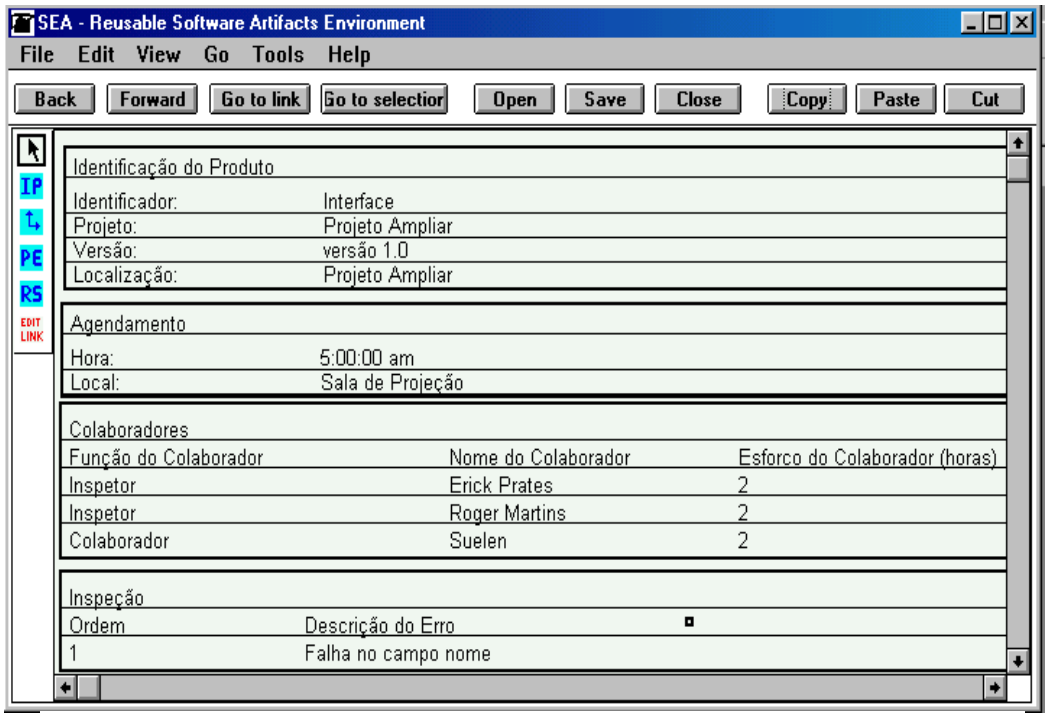


Figura 15 – Documento de Inspeção de Interface – atividade inspecionar interface

A figura 16 apresenta a atividade inspecionar interface realizada, ou seja, verificada sua consistência, e definida como congelada, isto é, *frozen* - concluída.

A cor verde na atividade inspecionar interface só é possível, após o documento de inspeção de interface, ter passado pelos procedimentos de análise de consistência como demonstraram as figuras 7, 8 e 9.

Lembra-se que a partir do momento em que o documento referenciado pela atividade em questão é passado pela análise de consistência, é validado e congelado, a atividade é automaticamente definida para o estado de realizada., representada pela cor verde.

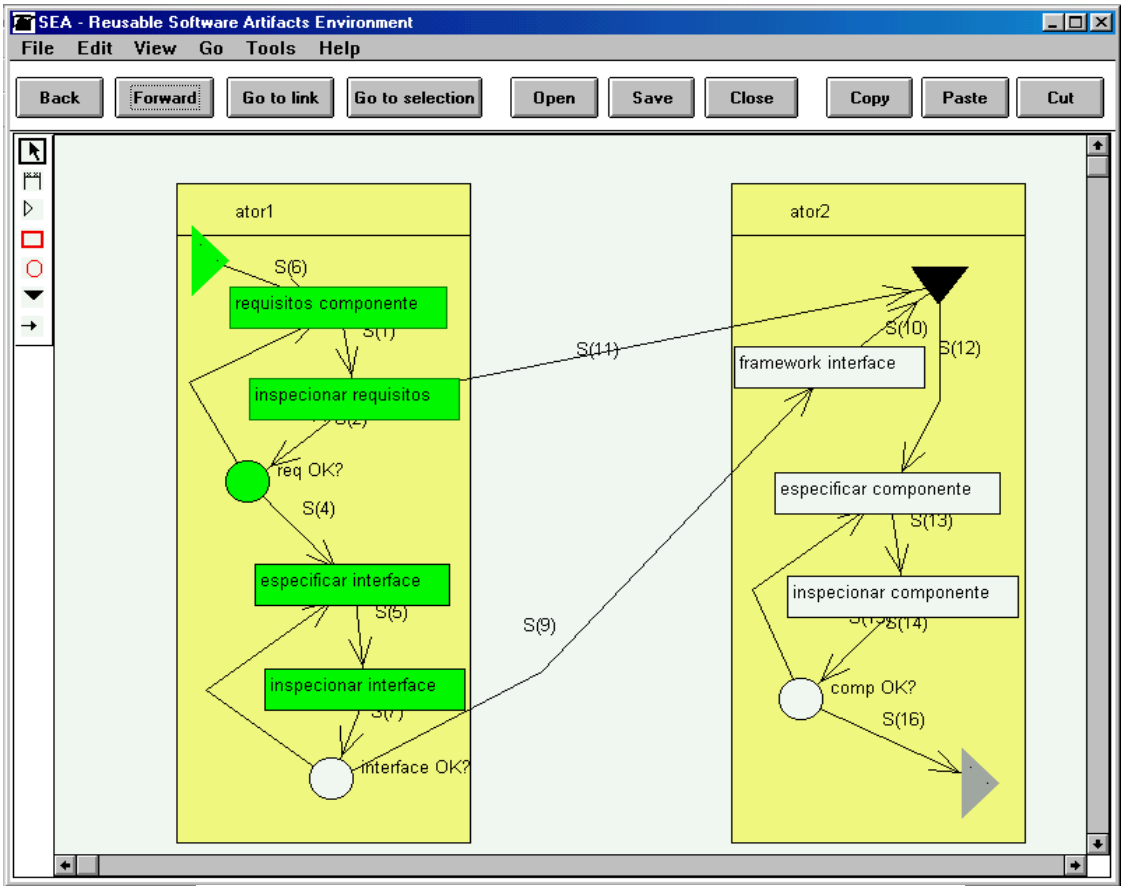


Figura 16 – Atividade inspecionar interface, realizada

Após o término da atividade inspecionar interface, uma decisão deve ser tomada para a continuação das atividades.

A figura 17 apresenta o símbolo que permite que uma decisão seja tomada após a realização de determinada atividade.

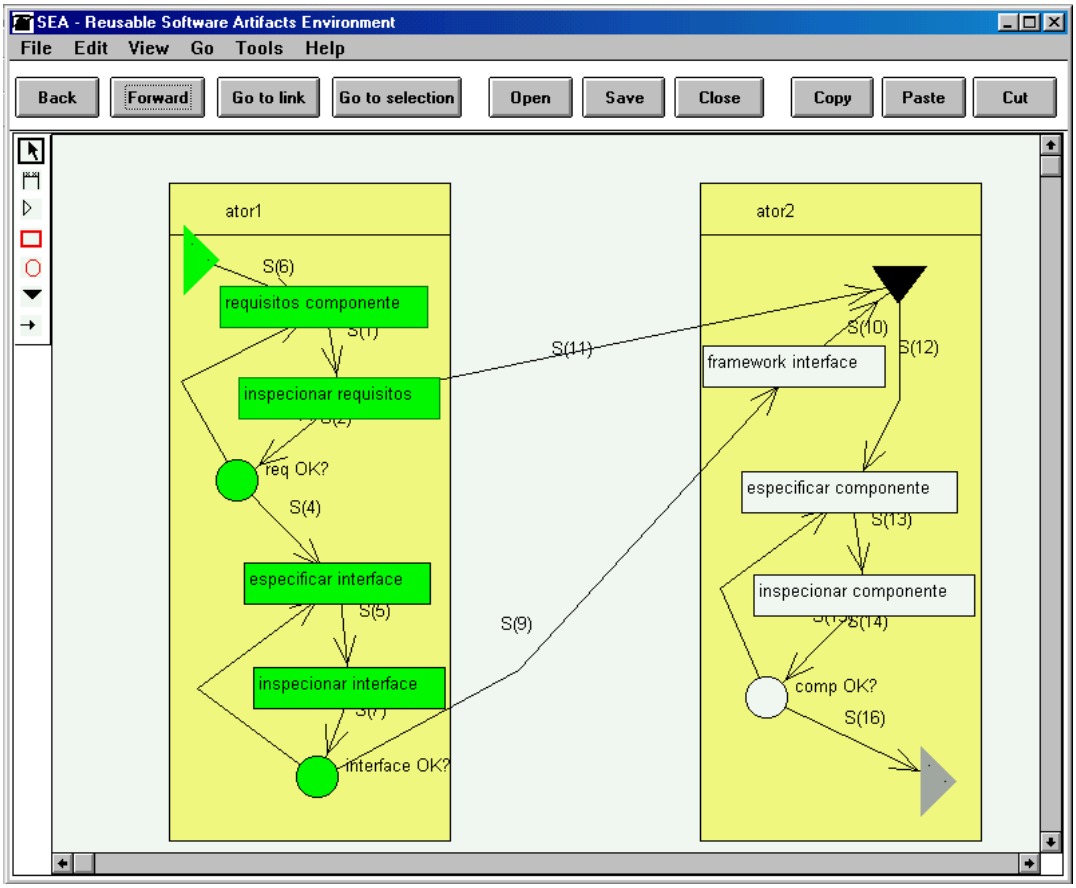


Figura 17 – Tomada de decisão

No caso da figura 17, o ator (responsável pela atividade) poderá refazer a atividade desde a atividade especificar interface caso a mesma não esteja “ok”, ou passar para a próxima atividade de acordo com a decisão tomada, que pode ser *framework* interface, ou seja, desenvolver a interface de um *framewok*. Para isso uma nova especificação, referente à criação da interface de um *framework* deve ser criada no ambiente, e com isso, estabelecer um *link* entre a atividade em questão e a especificação (ou modelo) que contém tal interface.

Na figura 18 é apresentada a atividade *framework* interface em andamento, devido a cor estar em azul. Tal atividade pertence a outra especificação que não a Estrutura de Documentos Textuais, por isso todos os passos não estão descritos, mas o procedimento de análise de consistência e “congelamento” do documento é o mesmo, com critérios de verificação de consistência específicos.

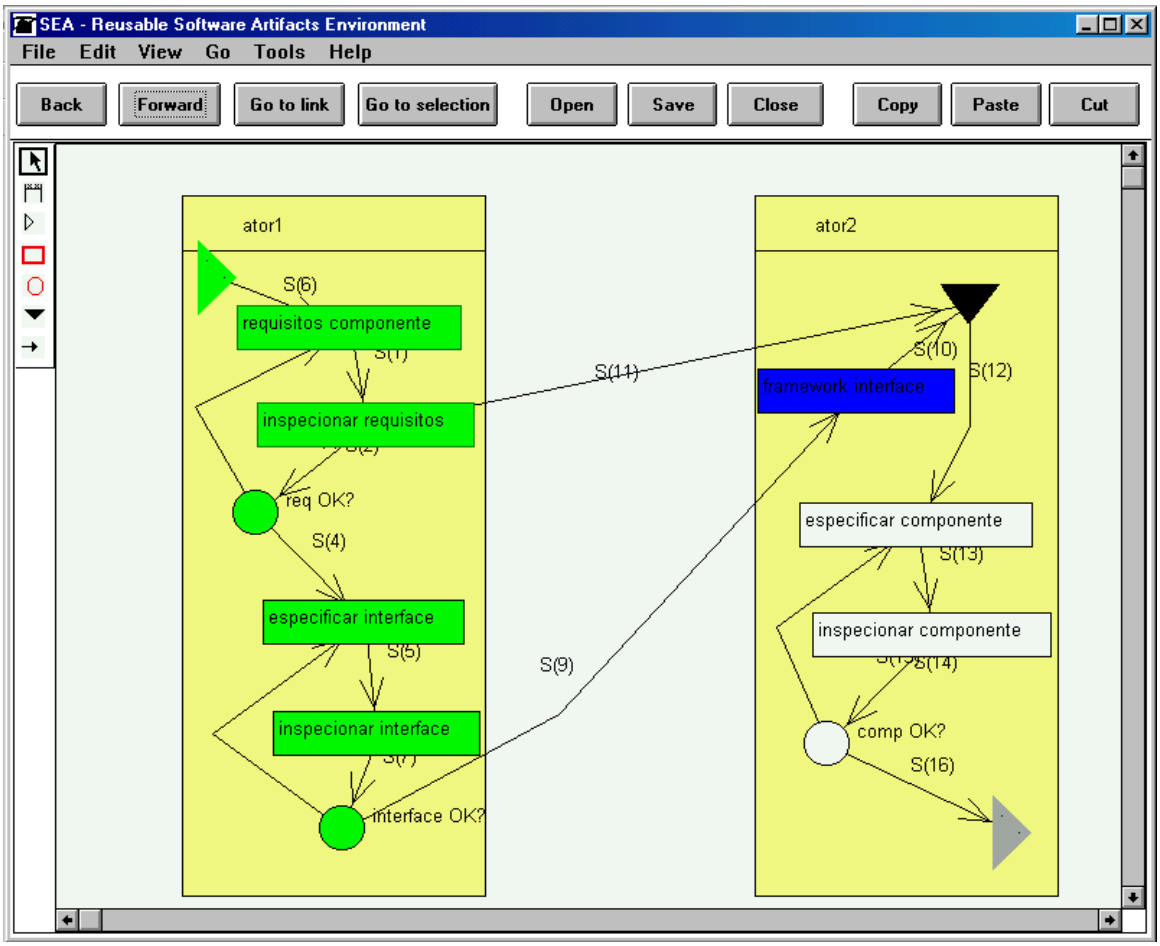


Figura 18 – Atividade *framework* interface, em andamento

A partir do momento em que a atividade *framework* interface for validada e definida como *frozen* - concluída, ela automaticamente passará ao estado de congelada representada no projeto de *workflow*, ou seja, atividade concluída apresentada na figura 19.

Concluída a atividade referente a desenvolver uma interface de *framework*, pode-se iniciar a próxima atividade – especificar componente como é apresentada na figura 20. Na figura 20 a atividade especificar componente deve, primeiramente, ser criada na especificação do ambiente SEA referente às atividades relacionados a criação de componentes, e linkada com essa atividade no projeto de *workflow*.

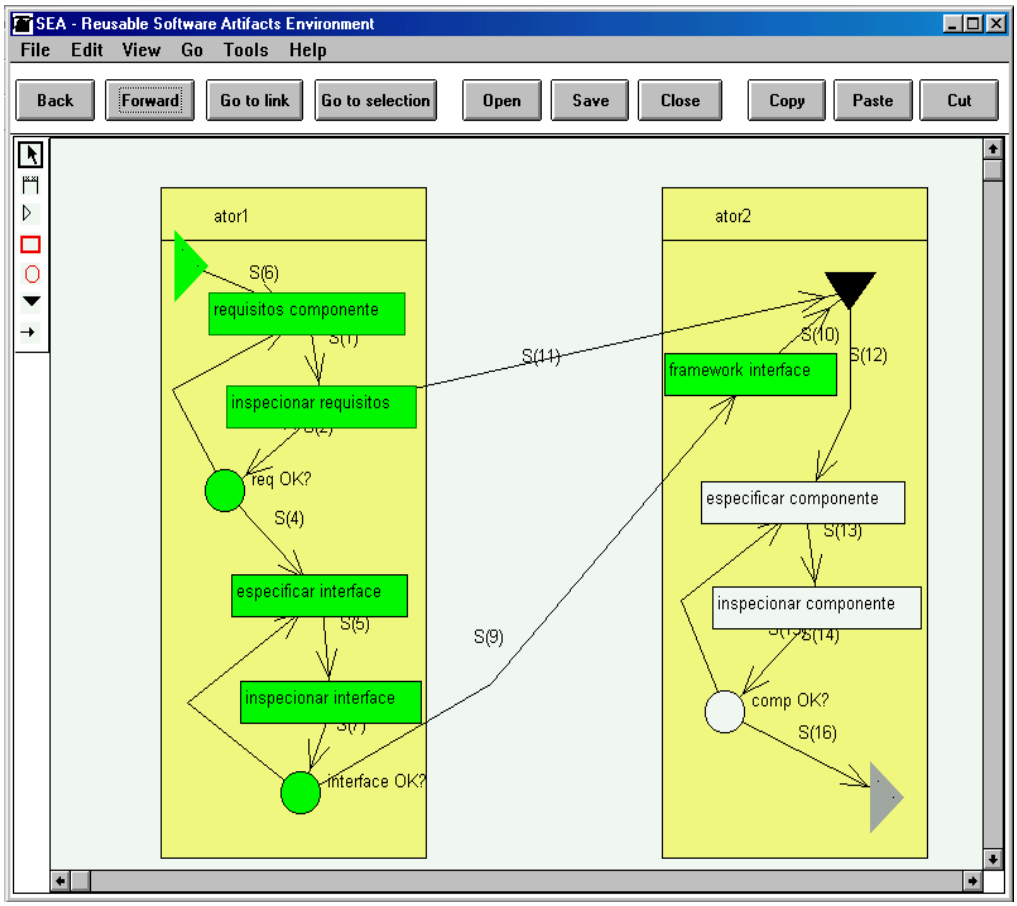


Figura 19 – Atividade *framework* interface, concluída

Na figura 20, a atividade especificar componente já começou a ser desenvolvida, mas está no estado “em andamento”. Isto quer dizer que na especificação que permite criar um documento deve ter sido definida como “em andamento”.

Após a atividade ter sido realizada na especificação a qual pertence, a mesma é validade e logo deve ser definida como “*frozen*” – “congelada” como apresenta a figura 21.

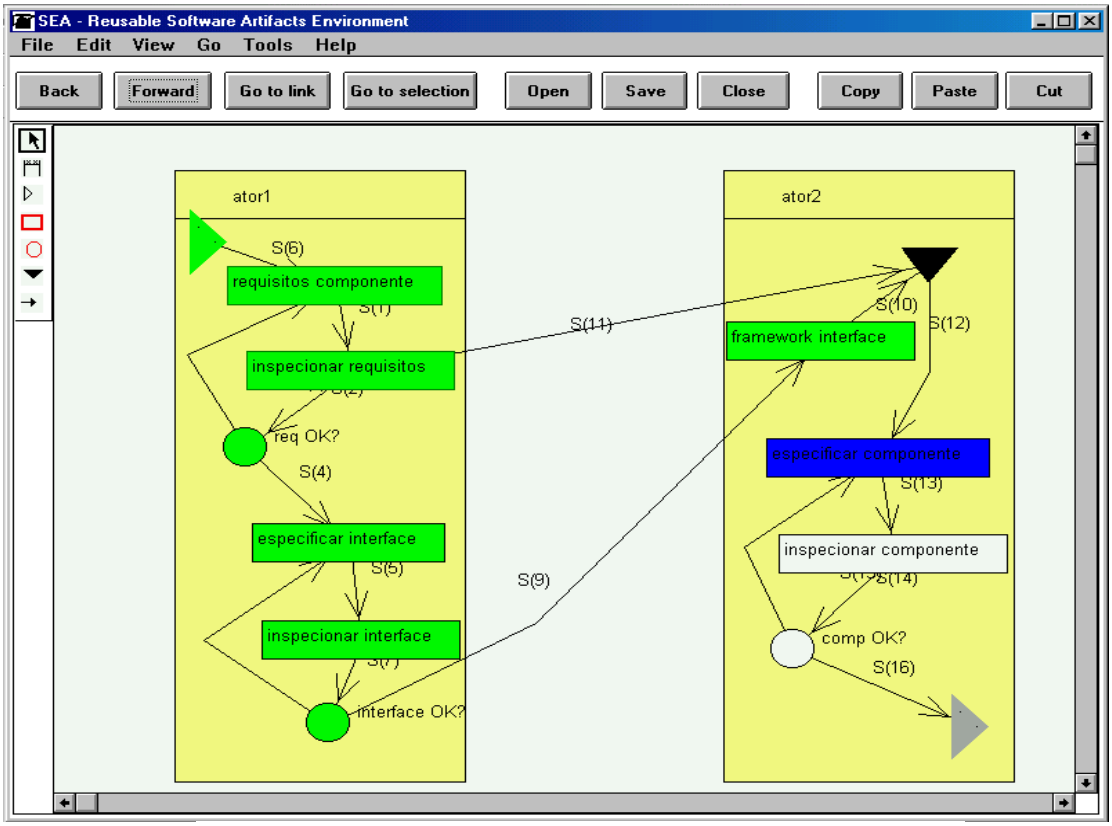


Figura 20– Atividade especificar componente, em andamento

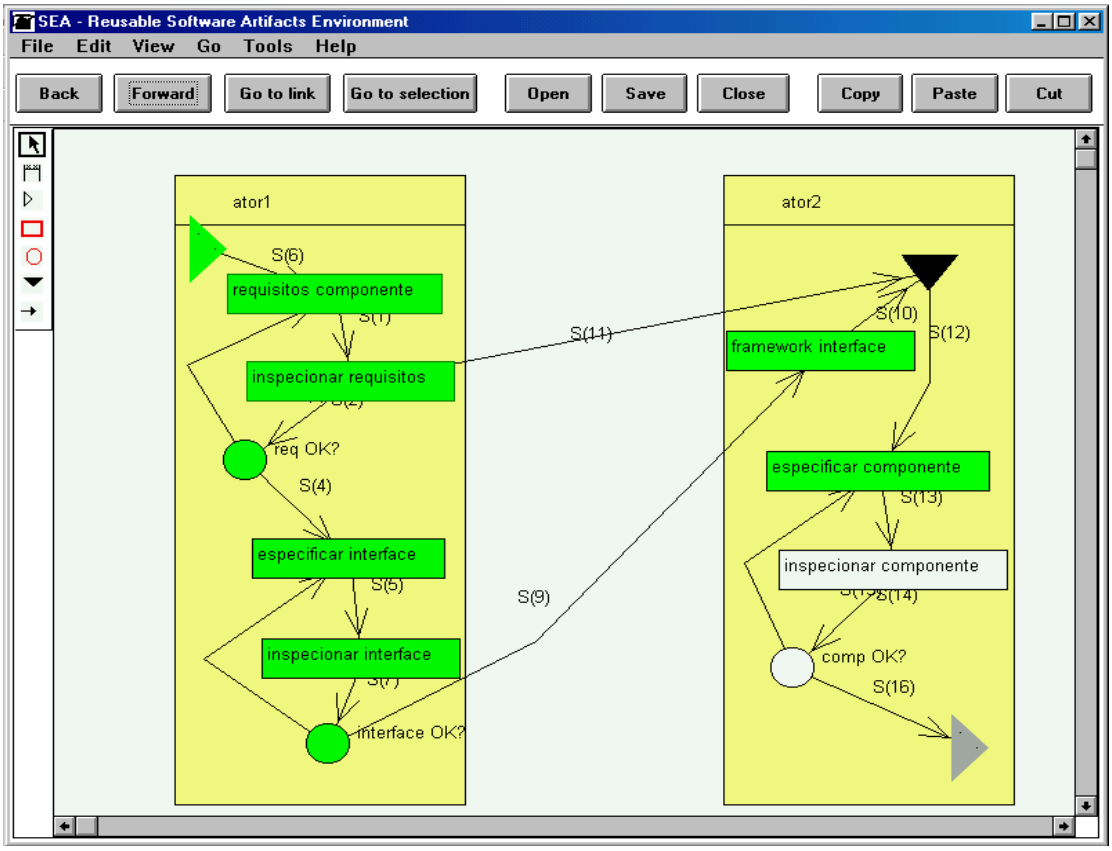


Figura 21 – Atividade especificar componente, realizada

Concluída a atividade especificar componente, representada na figura 21, pode-se iniciar a atividade inspecionar componente. Para isso deve-se, primeiramente, criar uma especificação responsável por permitir a edição de documento desse tipo, que é a Especificação Estrutura de Documentos Textuais (como na figura 3) e, então, criar um *link* entre a atividade em questão e o documento responsável por essa atividade, bem como deve ser definida a especificação à qual pertence esse documento.

Para que a atividade inspecionar componente possa ser identificada "em andamento" ou "congelada", o documento deve ter sido preenchido parcialmente ou totalmente, e, ainda, validado e ter seu estado definido como "em andamento", sem a verificação de consistência, ou concluído, com a verificação.

Definido como "em andamento" o documento de inspeção do componente, no Projeto de *Workflow*, o gerenciador automaticamente reconhece o seu estado e define a cor correspondente, (figura 22) desde que o documento de inspeção tenha sido parcialmente preenchido e definido como *frozen* como apresenta a figura 23.

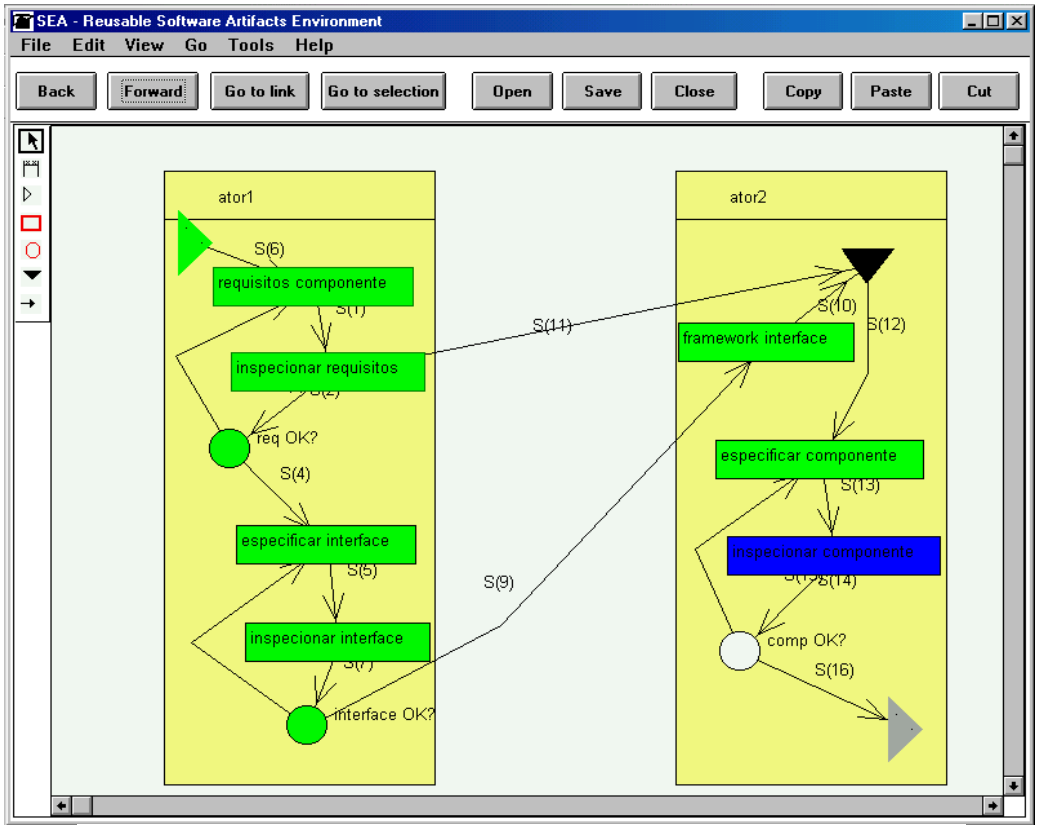


Figura 22 – Atividade "inspecionar componente", em andamento

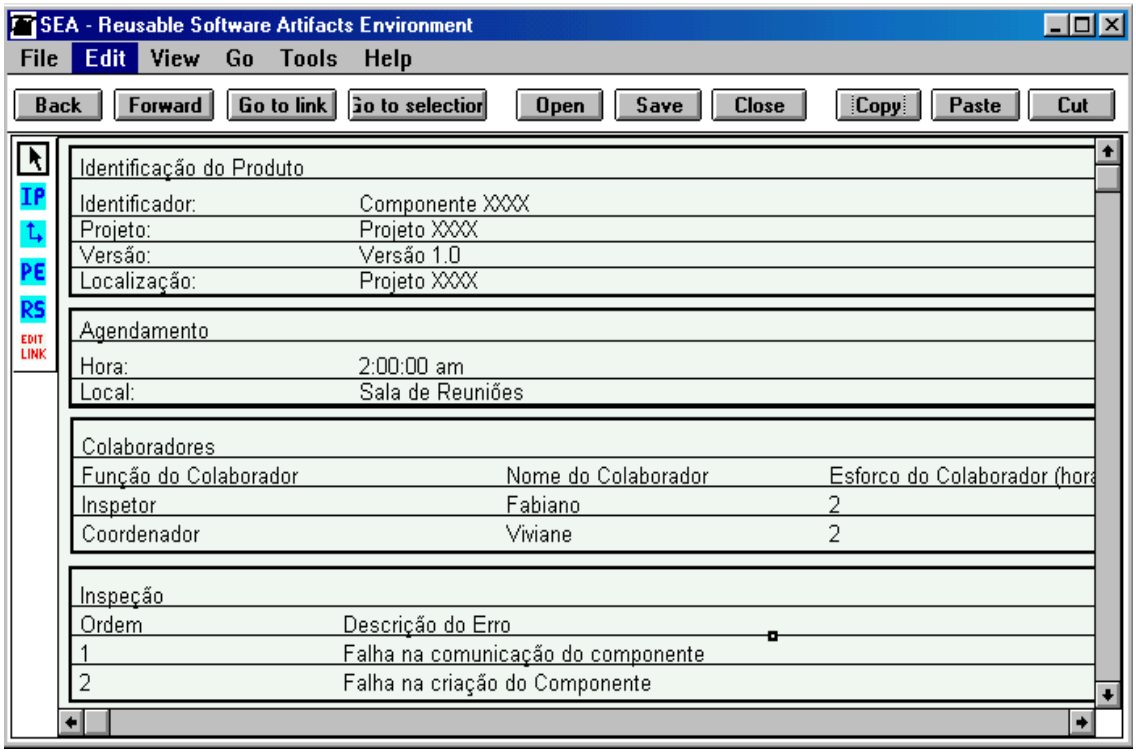


Figura 23 – Documento Inspeção parcialmente preenchido – atividade inspecionar componente

Para que a tal atividade seja definida no Projeto de *Workflow* como realizada, deve, primeiramente, concluir o preenchimento das informações no documento de inspeção do componente, analisar a consistência do documento e defini-la como "*Frozen* – congelada", como apresenta a figura 24 e iniciar o procedimento de análise de consistência apresentado pelas figuras 7, 8 e 9, selecionando a opção "*load Tool*..."(figura 24).

Após a validação do documento, automaticamente o gerenciador de *workflow* reconhece o estado da atividade referente ao documento como realizada, representada pela cor verde, apresentada na figura 25.

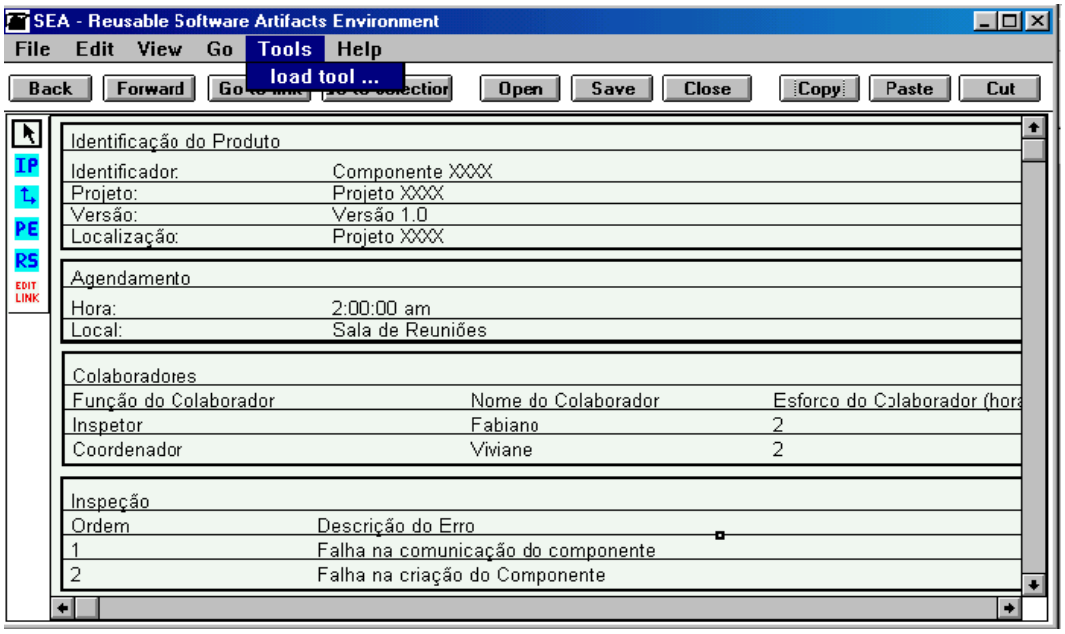


Figura 24 – Documento De Inspeção da atividade inspecionar componente, realizada

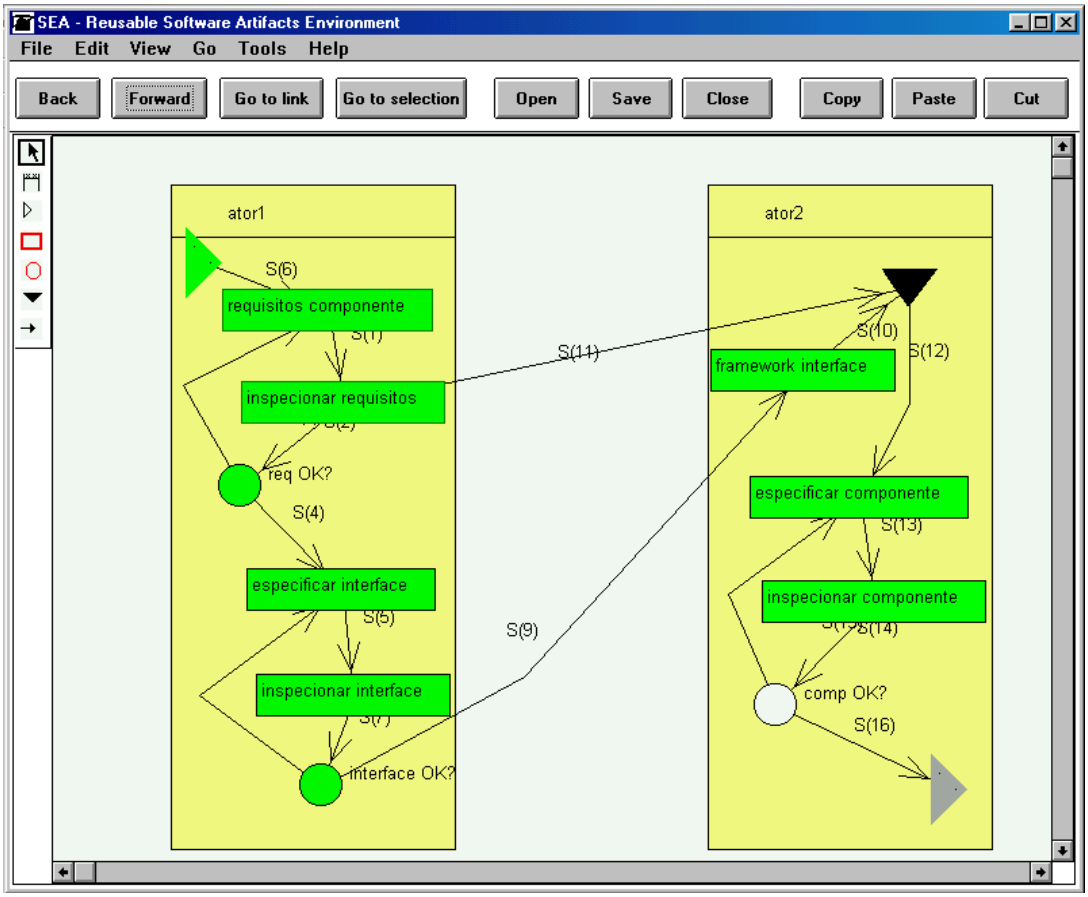


Figura 25 – Atividade inspecionar componente, realizada

Concluída a atividade inspecionar componente, pode-se tomar uma decisão, mas como as duas atividades anteriores estão ok e validadas, a decisão tomada é terminar o processo definido no Projeto de *Workflow*, como apresenta a figura 26.

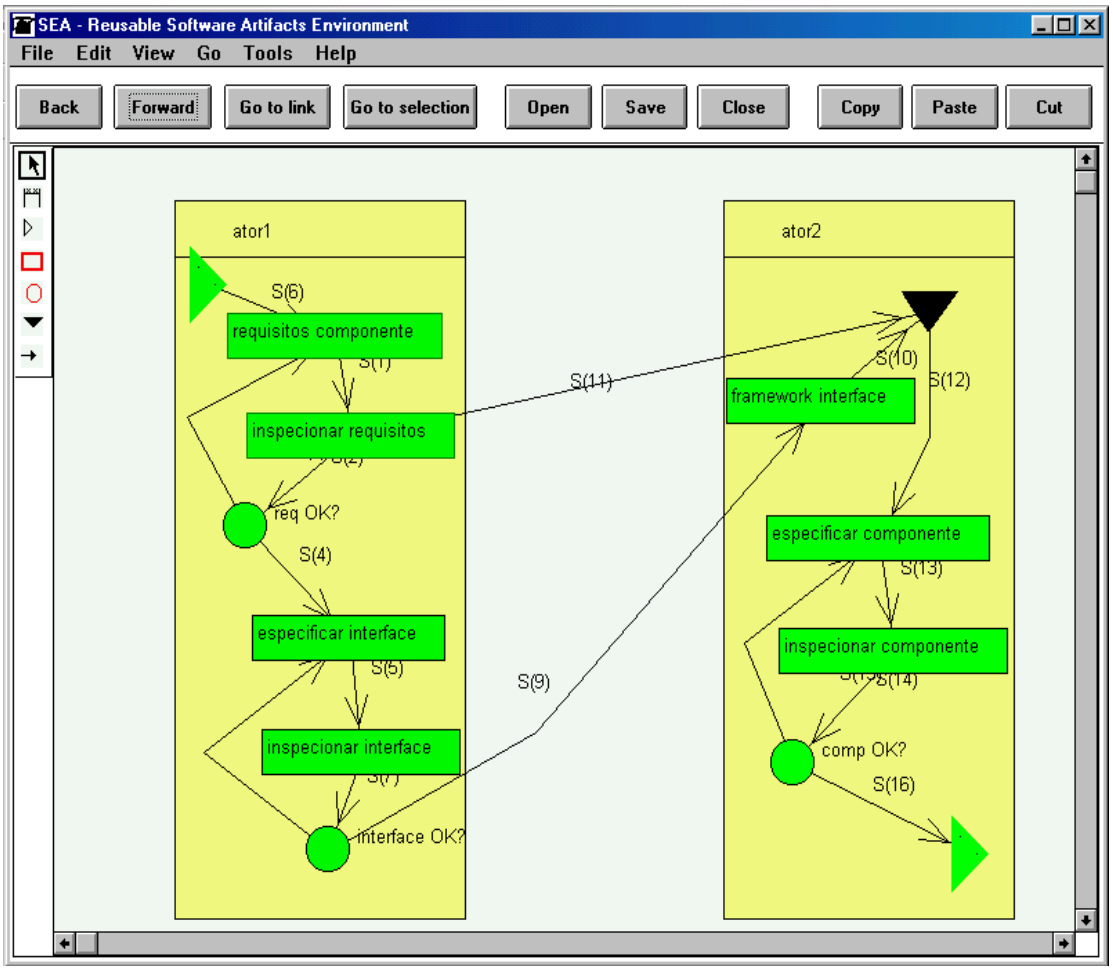


Figura 26 – Projeto de *Workflow* concluído

A figura 26 apresentou a concretização das atividades realizadas no ambiente SEA, propostas no exemplo do projeto de *Workflow*.

Neste exemplo tentou-se demonstrar a relevância deste trabalho com uma estrutura de documentos textuais num ambiente de desenvolvimento de *software*, embora nem todos os documentos necessários em um processo de desenvolvimento de *software* tenham sido implementados. Acredita-se que o objetivo principal do trabalho foi atingido: Criar um ensaio de documentos textuais estruturados que pudesse ser disponibilizado num ambiente de desenvolvimento de *software*, sendo o SEA o ambiente em estudo.